MATTHEW D. POWERS (Bar No. 104795)
matthew.powers@weil.com
EDWARD R. REINES (Bar No. 135960)
edward.reines@weil.com
JEFFREY G. HOMRIG (Bar No. 215890)
jeffrey.homrig@weil.com
JILL J. HO (Bar No. 236349)
jill.ho@weil.com
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
Telephone: (650) 802-3000
Facsimile: (650) 802-3100

Attorneys for Defendant-Counterplaintiff
NetApp, Inc.

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

SUN MICROSYSTEMS, INC., a Delaware corporation, Plaintiff-Counterdefendant, v. NETWORK APPLIANCE, INC., a Delaware corporation, Defendant-Counterplaintiff.	Case No. C-08-1641 EDL **NETAPP'S ANSWER TO SUN MICROSYSTEMS, INC.'S COMPLAINT FOR PATENT INFRINGEMENT** **DEMAND FOR JURY TRIAL**

Defendant NetApp, Inc. (formerly Network Appliance, Inc.) responds to the Complaint of Plaintiff Sun Microsystems, Inc. ("Sun") as follows:

THE PARTIES AND THE NATURE OF THE ACTION

1. NetApp admits that it spent approximately $390 million on research and development during the last reported fiscal year. NetApp holds more than 250 United States patents, is a pioneer in the design of data storage systems marketed throughout the United States and abroad, and continues to innovate new advances in data storage technology. NetApp's patents cover a host of advanced features found in NetApp's award-winning Data ONTAP® software and Write Anywhere File Layout (WAFL®) filesystem. These include fundamental developments in filesystems, unified storage solutions, data consistency, data integrity, storage management, write allocation, read-only data images (Snapshots™), writeable clones, copy-on-write, RAID arrays and assimilation, file system image transfer, and lock management mechanisms. NetApp's patented features are demanded by customers the world over because they greatly enhance the performance, reliability and ease of use of data storage systems. NetApp admits that it has not open-sourced its DataONTAP® software. NetApp denies that it merely builds upon the innovations of others without contributing developments of its own. NetApp lacks sufficient knowledge or information to form a belief as to the truth of the remaining allegations of paragraph 1 of Sun's Complaint and, on that basis, denies them.

2. NetApp lacks sufficient knowledge or information to form a belief as to the truth of the allegations of paragraph 2 of Sun's Complaint and, on that basis, denies them.

3. NetApp admits that on or about January 28, 2008, NetApp acquired Onaro, Inc., including Onaro's storage service management software.

4. NetApp denies the allegations of paragraph 4 of Sun's Complaint.

5. NetApp lacks sufficient knowledge or information to form a belief as to the truth of the allegations of paragraph 5 of Sun's Complaint and, on that basis, denies them.

6. NetApp admits the allegations of paragraph 6 of Sun's Complaint.

JURISDICTION

7. NetApp admits that this Court has subject matter jurisdiction over Sun's patent infringement claims pursuant to 28 U.S.C. §§ 1331 and 1338.

8. NetApp admits the allegations of paragraph 8 of Sun's Complaint.

VENUE

9. NetApp admits that venue is proper in this district. Except as expressly admitted, NetApp denies the allegations of paragraph 9 of Sun's Complaint.

INTRADISTRICT ASSIGNMENT

10. NetApp admits the allegations of paragraph 10 of Sun's Complaint.

THE PATENTS

11. NetApp admits that United States Patent No. 6,965,951 is entitled "Device Centric Discovery and Configuration for Fabric Devices"; that it states on its face that it was issued on November 15, 2005 to Hyon T. Kim; and that a copy of the '951 patent was attached as Exhibit A to Sun's Complaint. Except as expressly admitted, NetApp denies the allegations of paragraph 11 of Sun's Complaint.

12. NetApp admits that United States Patent No. 6,151,683 is entitled "Rebuilding Computer States Remotely"; that it states on its face that it was issued on November 21, 2000 to Michael J. Wookey; and that a copy of the '683 patent was attached as Exhibit B to Sun's Complaint. Except as expressly admitted, NetApp denies the allegations of paragraph 12 of Sun's Complaint.

13. NetApp admits that United States Patent No. 6,182,249 is entitled "Remote Alert Monitoring and Trend Analysis"; that it states on its face that it was issued on January 30, 2001 to Michael J. Wookey and Kevin L. Chu; and that a copy of the '249 patent was attached as Exhibit C to Sun's Complaint. Except as expressly admitted, NetApp denies the allegations of paragraph 13 of Sun's Complaint.

14. NetApp admits that United States Patent No. 6,484,200 is entitled "Distinguished Name Scoping System for Event Filtering"; that it states on its face that it was issued on November 19, 2002 to Rajeev Angal, Shivaram Bhat, Michael Roytman, and Subodh

Bapat; and that a copy of the '200 patent was attached as Exhibit D to Sun's Complaint. Except as expressly admitted, NetApp denies the allegations of paragraph 14 of Sun's Complaint.

FIRST CLAIM
(Infringement of the '951 Patent)

15. NetApp incorporates by reference its responses to the allegations of paragraphs 1 through 14 of Sun's Complaint.

16. NetApp denies the allegations of paragraph 16 of Sun's Complaint.

17. NetApp denies the allegations of paragraph 17 of Sun's Complaint.

18. NetApp admits that Sun informed NetApp of the existence of this patent by letter dated February 15, 2008, but denies that it infringes the '951 patent. Except as expressly admitted, NetApp denies the allegations of paragraph 18 of Sun's Complaint.

19. NetApp denies the allegations of paragraph 19 of Sun's Complaint.

20. NetApp denies the allegations of paragraph 20 of Sun's Complaint.

21. NetApp admits that this case is exceptional within the meaning of 35 U.S.C. § 285. NetApp is entitled to an award of attorneys' fees. Except as expressly admitted, NetApp denies the allegations of paragraph 21 of Sun's Complaint.

SECOND CLAIM
(Infringement of the '683 Patent)

22. NetApp incorporates by reference its responses to the allegations of paragraphs 1 through 14 of Sun's Complaint.

23. NetApp denies the allegations of paragraph 23 of Sun's Complaint.

24. NetApp denies the allegations of paragraph 24 of Sun's Complaint.

25. NetApp denies the allegations of paragraph 25 of Sun's Complaint.

26. NetApp denies the allegations of paragraph 26 of Sun's Complaint.

27. NetApp denies the allegations of paragraph 27 of Sun's Complaint.

28. NetApp admits that this case is exceptional within the meaning of 35 U.S.C. § 285. NetApp is entitled to an award of attorneys' fees. Except as expressly admitted, NetApp denies the allegations of paragraph 28 of Sun's Complaint.

THIRD CLAIM
(Infringement of the '249 Patent)

29. NetApp incorporates by reference its responses to the allegations of paragraphs 1 through 14 of Sun's Complaint.

30. NetApp denies the allegations of paragraph 30 of Sun's Complaint.

31. NetApp denies the allegations of paragraph 31 of Sun's Complaint.

32. NetApp denies the allegations of paragraph 32 of Sun's Complaint.

33. NetApp denies the allegations of paragraph 33 of Sun's Complaint.

34. NetApp denies the allegations of paragraph 34 of Sun's Complaint.

35. NetApp admits that this case is exceptional within the meaning of 35 U.S.C. § 285. NetApp is entitled to an award of attorneys' fees. Except as expressly admitted, NetApp denies the allegations of paragraph 35 of Sun's Complaint.

FOURTH CLAIM
(Infringement of the '200 Patent)

36. NetApp incorporates by reference its responses to the allegations of paragraphs 1 through 14 of Sun's Complaint.

37. NetApp denies the allegations of paragraph 37 of Sun's Complaint.

38. NetApp denies the allegations of paragraph 38 of Sun's Complaint.

39. NetApp denies the allegations of paragraph 39 of Sun's Complaint.

40. NetApp denies the allegations of paragraph 40 of Sun's Complaint.

41. NetApp denies the allegations of paragraph 41 of Sun's Complaint.

42. NetApp admits that this case is exceptional within the meaning of 35 U.S.C. § 285. NetApp is entitled to an award of attorneys' fees. Except as expressly admitted, NetApp denies the allegations of paragraph 42 of Sun's Complaint.

NETAPP'S RESPONSE TO SUN'S PRAYER FOR RELIEF

43. NetApp denies that Sun is entitled to any of the relief sought in its prayer for relief against NetApp, its agents, employees, representatives, successors and assigns, and those acting in privity or concert with NetApp. NetApp has not infringed, and is not infringing,

either directly or indirectly under section 271, any valid claim of any Sun Patents-in-Suit (U.S. Patent Nos. 6,965,951 ("the '951 patent"), 6,151,683 ("the '683 patent"), 6,182,249 ("the '249 patent"), and 6,484,200 ("the '200 patent")), either literally or under the doctrine of equivalents. Moreover, the Sun Patents-in-Suit are invalid and/or unenforceable. Sun is not entitled to recover damages, injunctive relief, costs, fees, interest, or any other type of recovery from NetApp. Sun's prayer should, therefore, be denied in its entirety and with prejudice, and Sun should take nothing. NetApp asks that judgment be entered for NetApp and that this action be found to be an exceptional case entitling NetApp to be awarded attorneys' fees in defending against Sun's Complaint, together with such other and further relief the Court deems appropriate.

AFFIRMATIVE DEFENSES

FIRST AFFIRMATIVE DEFENSE
(Failure to state a claim)

One or more of Sun's claims fails to state a claim upon which relief can be granted.

SECOND AFFIRMATIVE DEFENSE
(Invalidity)

NetApp alleges on information and belief that one or more claims of each of the Sun Patents-in-Suit is invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including without limitation those of sections 102, 103, and/or 112.

THIRD AFFIRMATIVE DEFENSE
(Unenforceability – Inequitable Conduct)

On information and belief, certain Sun Patents-in-Suit are unenforceable under the doctrine of inequitable conduct, as follows:

'951 Patent

On information and belief, prior to the issuance of the '951 patent, the named inventor and/or others substantively involved in prosecuting the applications leading to the '951 patent (collectively, "the applicants of the '951 patent") were aware of information material to the patentability of the claims of the '951 patent, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office ("the USPTO").

The material information that was withheld, concealed, and/or mischaracterized includes the identity of additional individuals who should have been named as inventors of the '951 patent. Specifically, on information and belief, the applicants of the '951 patent deliberately concealed and excluded inventors of two features claimed in the '951 patent.

The '951 patent claims persistent repository and node creation for fabric devices. Both these features are claimed as independent inventions in U.S. Patent Nos. 7,171,474 and 7,200,646, filed prior to the filing of the '951 patent. The '474 and '646 patents have the same assignee, Sun Microsystems, Inc., and name the same inventors: Shanthaveeraiah Sunil, Rastogi Aseem and Rao Raghavendra. None of the inventors named on the '474 and '646 patents is named as an inventor of the '951 patent, despite the fact that the '951 patent claims the features claimed in the '474 and '646 patents.

On information and belief, the applicants of the '951 patent concealed from the USPTO the existence of these additional inventors with the intent to deceive the USPTO.

'683 Patent

On information and belief, prior to the issuance of the '683 patent, the named inventors and/or others substantively involved in prosecuting the applications leading to the '683 patent (collectively, "the applicants of the '683 patent") were aware of information material to the patentability of the claims of the '683 patent, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office ("the USPTO"). The withheld information includes, but is not limited to, K. McCloghrie, IETF Request for Comments 1066, entitled "Management Information Base for Network Management of TCP/IP-based Internets," August 1988, available at http://www.ietf.org/rfc/rfc1066.txt?number=1066 ("the RFC on MIB"), and Subodh Bapat, OSI Management Information Base Implementation, IEEE/IFIP Symposium on Integrated Network Management II, 817, 831 (1991) ("the MIB article"). The author of the MIB article, Subodh Bapat, is a Distinguished Engineer and Vice President at Sun Microsystems. In addition, the RFC on MIB (which was published in 1988 and became a standard by which most TCP-IP devices are managed) was well known by those skilled

in the art at the time the application for the '683 patent was filed. On information and belief, one or more of the applicants, including Michael Wookey, a Distinguished Engineer and Chief Technology Officer at Sun Microsystems, Inc., was aware of the RFC on MIB reference and the MIB article before the application for the '683 patent was filed.

On information and belief, the applicants of the '683 patent repeatedly failed to identify these references to the examiner and made repeated representations during prosecution that, in light of their knowledge of these references, at least one of the applicants knew to be false. For example, when responding to various rejections, the applicants of the '683 patent claimed that the prior art does not teach hierarchical representation of a computer system in a static tree structure. Yet, anyone skilled in the art would understand that the RFC on MIB teaches representation of a computer system in a tree structure and was therefore highly material to the prosecution of the '683 patent. Moreover, the MIB article likewise includes extensive discussion of the use of Object Oriented models, "which are essentially tree structures" to model system management information, and was therefore also highly material to the prosecution of the '683 patent. MIB article, at 818 and 824.

On information and belief, the applicants of the '683 patent withheld this material information and mischaracterized the scope and content of the prior art with the intent to deceive the USPTO.

'249 Patent

On information and belief, prior to the issuance of the '249 patent, the named inventors and/or others substantively involved in prosecuting the applications leading to the '249 patent (collectively, "the applicants of the '249 patent") were aware of information material to the patentability of the claims of the '249 patent, but withheld, concealed, and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office ("the USPTO"). The withheld information includes, but is not limited to, K. McCloghrie, IETF Request for Comments 1066, entitled "Management Information Base for Network Management of TCP/IP-based Internets," August 1988, available at http://www.ietf.org/rfc/rfc1066.txt?number=1066

("the RFC on MIB"), and Subodh Bapat, OSI Management Information Base Implementation, IEEE/IFIP Symposium on Integrated Network Management II, 817, 831 (1991) ("the MIB article"). The author of the MIB article, Subodh Bapat, is a Distinguished Engineer and Vice President at Sun Microsystems. In addition, the RFC on MIB (which was published in 1988 and became a standard by which most TCP-IP devices are managed) was well known by those skilled in the art at the time the application for the '249 patent was filed. On information and belief, one or more of the applicants, including Michael Wookey, a Distinguished Engineer and Chief Technology Officer at Sun Microsystems, Inc., was aware of the RFC on MIB reference and the MIB article before the application for the '249 patent was filed.

On information and belief, the applicants of the '249 patent repeatedly failed to identify these references to the examiner and made repeated representations during prosecution that, in light of their knowledge of these references, at least one of the applicants knew to be false. For example, when responding to various rejections, the applicants of the '249 patent claimed that the prior art does not teach hierarchical representation of a computer system in a modifiable tree structure. Yet, anyone skilled in the art would understand that the RFC on MIB teaches hierarchical representation of a computer system in a tree structure and was therefore highly material to the prosecution of the '249 patent. Moreover, the MIB article likewise includes extensive discussion of the use of Object Oriented models, "which are essentially tree structures" to model system management information. MIB article, at 818 and 824. In addition, section 5.2 of the MIB article describes a "MIB Schema modification" and the article expressly teaches a modifiable tree structure. *See id*. at 823, 828.

On information and belief, the applicants of the '249 patent withheld this material information and mischaracterized the scope and content of the prior art with the intent to deceive the USPTO.

'200 Patent

On information and belief, prior to the issuance of the '200 patent, the named inventors and/or others substantively involved in prosecuting the applications leading to the '200

patent (collectively, "the applicants of the '200 patent") were aware of information material to the patentability of the claims of the '200 patent, but withheld, concealed and/or mischaracterized that information with the intent to deceive the U.S. Patent and Trademark Office ("the USPTO"). The withheld information includes, but is not limited to, U.S. Patent Nos. 5,355,484 and 5,446,878.

One or more of the applicants of the '200 patent knew about U.S. Patent No. 5,355,484 before the '200 patent issued, because this patent was cited in a co-pending patent application by one or more of the same inventors, including without limitation the application that led to U.S. Patent No. 6,298,378 B1.

One or more of the applicants of the '200 patent knew about U.S. Patent No. 5,446,878 before the '200 patent issued, because this patent was cited in a co-pending patent application by one or more of the same inventors, including without limitation the application that led to U.S. Patent No. 6,542,932.

Both of these references describe monitoring events that occur at particular layers of a computer system, or with respect to particular components in a computer system, and thus were highly material to the patentability of the claims of the '200 patent, but were not disclosed to the USPTO during the prosecution of the '200 patent. On information and belief, the applicants of the '200 patent withheld these references from the USPTO with the intent to deceive the USPTO.

FOURTH AFFIRMATIVE DEFENSE
(Laches)

On information and belief, the relief that Sun seeks in its Complaint is barred in whole or in part by the doctrine of laches. On information and belief, Sun was aware of NetApp's (previously, Onaro's) products or was in possession of such facts and information as would lead a person of reasonable diligence to discover them well before Sun actually filed suit. Sun's unreasonable and inexcusable delay in filing suit has resulted in prejudice and injury to NetApp.

FIFTH AFFIRMATIVE DEFENSE
(Patent Misuse)

On information and belief, one or more of the Sun Patents-in-Suit is unenforceable under the doctrine of patent misuse.

COUNTERCLAIMS

NetApp, for its Counterclaims against Sun, alleges as follows:

THE PARTIES AND THE NATURE OF THIS ACTION

1. NetApp incorporates by reference paragraphs 1-14 of its Answer and its Affirmative Defenses as though set forth fully herein.

2. Defendant Sun uses NetApp's patented inventions in its data processing systems and related software, primarily Sun's Zettabyte File System ("ZFS"), without authorization from NetApp and thus directly infringes United States Patent Nos. 6,516,351, 7,293,097, 7,293,152, and 7,328,305, collectively referred to as the "NetApp Patents-in-Suit." Defendant Sun actively induces infringement and contributes to the infringement by others of NetApp's patents by urging and enabling others to adopt and distribute infringing technology in their products, and by distributing ZFS in or for data processing systems.

3. Sun unfairly distributes ZFS technology to others to induce others to adopt and distribute the infringing technology in their products without informing them of NetApp's applicable patents.

4. NetApp brings these counterclaims to halt Sun's willful infringement of its rights under the patent laws of the United States (Title 35, United States Code, section 271 et seq.). NetApp seeks damages for Sun's infringement and an injunction restraining Sun from continuing to use NetApp's patented inventions without permission.

JURISDICTION AND VENUE

5. NetApp is a Delaware corporation with its principal place of business at 495 E. Java Drive, Sunnyvale, California 94089.

6. Upon information and belief, Sun is a Delaware corporation with a principal place of business at 4150 Network Circle, Santa Clara, California 95054.

7. This Court has personal jurisdiction over Sun because it maintains its

principal place of business within the Northern District of California and it has committed acts of patent infringement within the State of California by making, using, selling, offering to sell, and/or importing products that are covered by NetApp's patents.

8. This Court has subject matter jurisdiction over NetApp's patent infringement counterclaims under 28 U.S.C. §§ 1331 and 1338(a).

9. This Court has subject matter jurisdiction over NetApp's declaratory judgment counterclaims under 28 U.S.C. §§ 2201 and 2202. There is an actual controversy between NetApp and Sun regarding U.S. Patent Nos. 6,965,951 ("the '951 patent"), 6,151,683 ("the '683 patent"), 6,182,249 ("the '249 patent"), and 6,484,200 ("the '200 patent") (collectively, "the Sun Patents-in-Suit") because Sun asserted in its Complaint that NetApp infringes the Sun Patents-in-Suit.

10. An actual and justifiable controversy exists between NetApp and Sun as to whether the Sun Patents-in-Suit are infringed, valid, and enforceable against NetApp. Absent a declaration of non-infringement, invalidity, or unenforceability, Sun will continue to wrongfully assert the Sun Patents-in-Suit against NetApp, and thereby cause NetApp irreparable injury and damage.

11. Venue is proper in this judicial district under 28 U.S.C. §§ 1391 and 1400(b).

FIRST COUNTERCLAIM
(Infringement of the '351 Patent)

12. NetApp incorporates by reference the allegations of paragraphs 1 through 14 of its Answer and paragraphs 1 through 11 of these Counterclaims.

13. NetApp is the owner of U.S. Patent No. 6,516,351 ("the '351 patent"), entitled "Enforcing Uniform File-Locking for Diverse File-Locking Protocols." The '351 patent was duly and legally issued on February 4, 2003. A true and correct copy of the '351 patent is attached to these Counterclaims as Exhibit A.

14. On information and belief, Sun has been and is currently infringing one or more claims of the '351 patent, directly and/or indirectly, pursuant to 35 U.S.C. § 271, in

connection with certain of its products, services, methods and/or systems, including without limitation Sun's ZFS.

15. Unless enjoined, Sun will continue to infringe, directly and/or indirectly under section 271, the '351 patent.

16. Sun's conduct has caused, and unless enjoined will continue to cause, irreparable harm to NetApp. Pursuant to 35 U.S.C. § 283, NetApp is entitled to a permanent injunction against further infringement.

17. Pursuant to 35 U.S.C. § 284, NetApp is entitled to damages for Sun's infringement.

18. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

SECOND COUNTERCLAIM
(Infringement of the '097 Patent)

19. NetApp incorporates by reference the allegations of paragraphs 1 through 14 of its Answer and paragraphs 1 through 11 of these Counterclaims.

20. NetApp is the owner of U.S. Patent No. 7,293,097 ("the '097 patent"), entitled "Enforcing Uniform File-Locking for Diverse File-Locking Protocols." The '097 patent was duly and legally issued on November 6, 2007. A true and correct copy of the '097 patent is attached to these Counterclaims as Exhibit B.

21. On information and belief, Sun has been and is currently infringing one or more claims of the '097 patent, directly and/or indirectly, pursuant to 35 U.S.C. § 271, in connection with certain of its products, services, methods and/or systems, including without limitation Sun's ZFS.

22. Unless enjoined, Sun will continue to infringe, directly and/or indirectly under section 271, the '097 patent.

23. Sun's conduct has caused, and unless enjoined will continue to cause, irreparable harm to NetApp. Pursuant to 35 U.S.C. § 283, NetApp is entitled to a permanent injunction against further infringement.

24. Pursuant to 35 U.S.C. § 284, NetApp is entitled to damages for Sun's infringement.

25. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

THIRD COUNTERCLAIM
(Infringement of the '152 Patent)

26. NetApp incorporates by reference the allegations of paragraphs 1 through 14 of its Answer and paragraphs 1 through 11 of these Counterclaims.

27. NetApp is the owner of U.S. Patent No. 7,293,152 ("the '152 patent"), entitled "Consistent Logical Naming of Initiator Groups." The '152 patent was duly and legally issued on November 6, 2007. A true and correct copy of the '152 patent is attached to these Counterclaims as Exhibit C.

28. On information and belief, Sun has been and is currently infringing one or more claims of the '152 patent, directly and/or indirectly, pursuant to 35 U.S.C. § 271, in connection with certain of its products, services, methods and/or systems, including without limitation Sun's ZFS.

29. Unless enjoined, Sun will continue to infringe, directly and/or indirectly under section 271, the '152 patent.

30. Sun's conduct has caused, and unless enjoined will continue to cause, irreparable harm to NetApp. Pursuant to 35 U.S.C. § 283, NetApp is entitled to a permanent injunction against further infringement.

31. Pursuant to 35 U.S.C. § 284, NetApp is entitled to damages for Sun's infringement.

32. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

FOURTH COUNTERCLAIM
(Infringement of the '305 Patent)

33. NetApp incorporates by reference the allegations of paragraphs 1 through 14 of its Answer and paragraphs 1 through 11 of these Counterclaims.

34. NetApp is the owner of U.S. Patent No. 7,328,305 ("the '305 patent"), entitled "Dynamic Parity Distribution Technique." The '305 patent was duly and legally issued on February 5, 2008. A true and correct copy of the '305 patent is attached to these Counterclaims as Exhibit D.

35. On information and belief, Sun has been and is currently infringing one or more claims of the '305 patent, directly and/or indirectly, pursuant to 35 U.S.C. § 271, in connection with certain of its products, services, methods and/or systems, including without limitation Sun's ZFS.

36. Unless enjoined, Sun will continue to infringe, directly and/or indirectly under section 271, the '305 patent.

37. Sun's conduct has caused, and unless enjoined will continue to cause, irreparable harm to NetApp. Pursuant to 35 U.S.C. § 283, NetApp is entitled to a permanent injunction against further infringement.

38. Pursuant to 35 U.S.C. § 284, NetApp is entitled to damages for Sun's infringement.

39. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

FIFTH COUNTERCLAIM

(Declaratory Judgment Re: the '951 Patent)

40. NetApp incorporates by reference the allegations of paragraphs 1 through 21 of its Answer and paragraphs 1 through 11 of these Counterclaims.

41. Sun alleges in paragraph 11 of its Complaint that it is the assignee of the '951 patent, entitled "Device Centric Discovery and Configuration for Fabric Devices."

42. Sun asserted in its Complaint that NetApp infringes the '951 patent.

43. NetApp has not infringed, and is not infringing, either directly or indirectly under section 271, any valid claim of the '951 patent, either literally or under the doctrine of equivalents.

44. One or more claims of the '951 patent is invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including without limitation those of sections 102, 103, and/or 112.

45. The '951 patent is unenforceable for at least the reasons alleged in NetApp's affirmative defenses, which are incorporated herein by reference.

46. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

SIXTH COUNTERCLAIM

(Declaratory Judgment Re: the '683 Patent)

47. NetApp incorporates by reference the allegations of paragraphs 1 through 14 and 22 through 28 of its Answer and paragraphs 1 through 11 of these Counterclaims.

48. Sun alleges in paragraph 12 of its Complaint that it is the assignee of the '683 patent, entitled "Rebuilding Computer States Remotely."

49. Sun asserted in its Complaint that NetApp infringes the '683 patent.

50. NetApp has not infringed, and is not infringing, either directly or indirectly under section 271, any valid claim of the '683 patent, either literally or under the doctrine of equivalents.

51. One or more claims of the '683 patent is invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including without limitation those of sections 102, 103, and/or 112.

52. The '683 patent is unenforceable for at least the reasons alleged in NetApp's affirmative defenses, which are incorporated herein by reference.

53. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

SEVENTH COUNTERCLAIM

(Declaratory Judgment Re: the '249 Patent)

54. NetApp incorporates by reference the allegations of paragraphs 1 through 14 and 29 through 35 of its Answer and paragraphs 1 through 11 of these Counterclaims.

55. Sun alleges in paragraph 13 of its Complaint that it is the assignee of the '249 patent, entitled "Remote Alert Monitoring and Trend Analysis."

56. Sun asserted in its Complaint that NetApp infringes the '249 patent.

57. NetApp has not infringed, and is not infringing, either directly or indirectly under section 271, any valid claim of the '249 patent, either literally or under the doctrine of equivalents.

58. One or more claims of the '249 patent is invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including without limitation those of sections 102, 103, and/or 112.

59. The '249 patent is unenforceable for at least the reasons alleged in NetApp's affirmative defenses, which are incorporated herein by reference.

60. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

EIGHTH COUNTERCLAIM

(Declaratory Judgment Re: the '200 Patent)

61. NetApp incorporates by reference the allegations of paragraphs 1 through 14 and 36 through 42 of its Answer and paragraphs 1 through 11 of these Counterclaims.

62. Sun alleges in paragraph 14 of its Complaint that it is the assignee of the '200 patent, entitled "Distinguished Name Scoping System for Event Filtering."

63. Sun asserted in its Complaint that NetApp infringes the '200 patent.

64. NetApp has not infringed, and is not infringing, either directly or indirectly under section 271, any valid claim of the '200 patent, either literally or under the doctrine of equivalents.

65. One or more claims of the '200 patent is invalid for failure to meet the conditions of patentability of 35 U.S.C. § 101 et seq., including without limitation those of sections 102, 103, and/or 112.

66. The '200 patent is unenforceable for at least the reasons alleged in NetApp's affirmative defenses, which are incorporated herein by reference.

67. This case is exceptional and, pursuant to 35 U.S.C. § 285, NetApp is entitled to an award of attorneys' fees.

PRAYER FOR RELIEF

WHEREFORE, NetApp respectfully requests the following relief:

A. a judgment in favor of NetApp finding that Sun has infringed the '351, '097, '152, and '305 patents;

B. a permanent injunction enjoining Sun, its officers, agents, servants, employees, attorneys, and those persons in active concert or participation with any of them, from directly or indirectly infringing the '351, '097, '152, and '305 patents, including the distribution of any current or future versions of Sun's ZFS;

C. an accounting for damages resulting from Sun's infringement of the '351, '097, '152, and '305 patents;

D. an award of damages to compensate NetApp for Sun's infringement, pursuant to 35 U.S.C. § 284, said damages to be trebled because the infringement is and has been willful;

E. an assessment of pre-judgment and post-judgment interest and costs against Sun, together with an award of such interest and costs, in accordance with 35 U.S.C. § 284;

F. a declaration that NetApp has not infringed and is not infringing, directly or indirectly, any claims of the '951, '683, '249, and '200 patents;

G. a declaration that the claims of the '951, '683, '249, and '200 patents are invalid;

H. a declaration that the '951, '683, '249, and '200 patents are unenforceable;

I. a declaration that all Defendants and each of their officers, employees, agents, alter egos, attorneys, and any persons in active concert or participation with them be restrained and enjoined from further prosecuting or instituting any action against NetApp claiming that the '951, '683, '249, and '200 patents are infringed, valid, or enforceable, or from representing that NetApp's products or services, or that others' use thereof, infringe the '951, '683, '249, and '200 patents;

J. an award to NetApp of its attorneys' fees incurred in connection with this lawsuit pursuant to 35 U.S.C.§ 285;

K. costs incurred in bringing NetApp's counterclaims; and

L. such other and further relief as this Court may deem just and proper.

JURY DEMAND

Plaintiff respectfully requests a trial by jury, pursuant to Fed. R. Civ. P. 38(b), on all issues so triable.

Dated: May 19, 2008

WEIL, GOTSHAL & MANGES LLP

By: */s/ Edward R. Reines*
Edward R. Reines
Attorney for Defendant
NetApp, Inc.

CERTIFICATION OF INTERESTED ENTITIES OR PERSONS

Pursuant to Civil L.R. 3-16, the undersigned certifies that as of this date, other than the named parties and their shareholders, there is no such interest to report. Pursuant to Rule 7.1 of the Federal Rules of Civil Procedure, the undersigned certifies that: i) NetApp does not have a parent corporation, and ii) the following publicly-traded companies hold a ten percent or more share of NetApp's stock: Alliance Bernstein LP.

Dated: May 19, 2008

WEIL, GOTSHAL & MANGES LLP

By: */s/ Edward R. Reines*
Edward R. Reines
Attorney for Defendant
NetApp, Inc.

Exhibit A

(12) **United States Patent**
Borr

(10) **Patent No.: US 6,516,351 B2**
(45) **Date of Patent: *Feb. 4, 2003**

(54) **ENFORCING UNIFORM FILE-LOCKING FOR DIVERSE FILE-LOCKING PROTOCOLS**

(75) Inventor: **Andrea Borr**, Los Gatos, CA (US)

(73) Assignee: **Network Appliance, Inc.**, Sunnyvale, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/176,599**

(22) Filed: **Oct. 21, 1998**

(65) **Prior Publication Data**

US 2002/0019874 A1 Feb. 14, 2002

Related U.S. Application Data

(63) Continuation of application No. 08/985,398, filed on Dec. 5, 1997, now abandoned.

(51) **Int. Cl.**[7] **G06F 17/30**
(52) **U.S. Cl.** **709/229**; 707/1
(58) **Field of Search** 709/201, 203, 709/217, 219, 213, 229; 707/1, 7, 8, 9, 10, 103; 710/200

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,719,569 A 1/1988 Ludemann et al.

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP 0306244 A2 3/1989

(List continued on next page.)

OTHER PUBLICATIONS

"Mapping the VM text files to the AIX text files", IBM Technical Disclosure Bulletin., vol. 33, No. 2, Jul. 1990, p. 341 XP000123641, IBM Corp. New York., US ISSN: 0018–8689—the whole document.

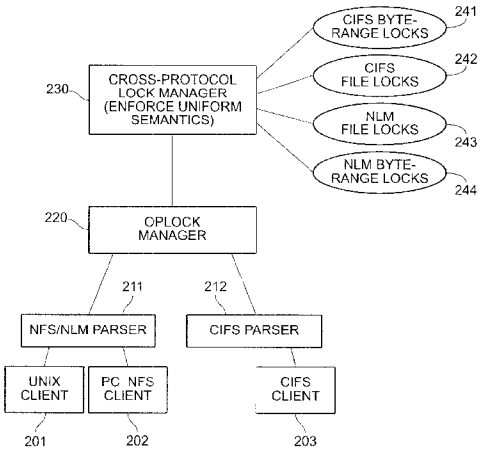
(List continued on next page.)

Primary Examiner—Le Hien Luu
Assistant Examiner—Bunjob Jaroenchonwanit
(74) *Attorney, Agent, or Firm*—Swernofsky Law Group PC

(57) **ABSTRACT**

The invention provides a method and system for correct interoperation of multiple diverse file server or file locking protocols, using a uniform multi-protocol lock management system. A file server determines, before allowing any client device to access data or to obtain a lock, whether that would be inconsistent with existing locks, regardless of originating client device or originating protocol for those existing locks. A first protocol enforces mandatory file-open and file-locking together with an opportunistic file-locking technique, while a second protocol lacks file-open semantics and provides only for advisory byte-range and file locking. Enforcing file-locking protects file data against corruption by NFS client devices. A CIFS client device, upon opening a file, can obtain an "oplock" (an opportunistic lock). When a client device issues a non-CIFS protocol request for the oplocked file, the file server sends an oplock-break message to the CIFS client device, giving the CIFS client device the opportunity to flush any cached write operations and possibly close the file. Allowing NFS and NLM requests to break oplocks ensures that file data remains available to NFS client devices simultaneously with protecting integrity of that file data. A CIFS client device can obtain a "change-monitoring" lock for a directory in the file system, so as to be notified by the file server whenever there is a change to that directory. The file server notes changes to the directory by both CIFS and non-CIFS client devices, and notifies those CIFS client devices with "change-monitoring" locks of those changes.

114 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

4,742,450	A		5/1988	Duvall et al.
4,780,821	A		10/1988	Crossley
4,825,354	A		4/1989	Agrawal et al. 364/200
4,887,204	A		12/1989	Johnson et al.
4,937,763	A		6/1990	Mott 364/550
4,984,272	A		1/1991	McIlroy et al. 380/25
5,008,786	A		4/1991	Thatte
5,043,876	A		8/1991	Terry
5,067,099	A		11/1991	McCown et al. 364/550
5,113,442	A		5/1992	Moir 380/25
5,144,659	A		9/1992	Jones 380/4
5,202,983	A		4/1993	Orita et al. 395/600
5,222,217	A		6/1993	Blount et al. 395/325
5,261,051	A		11/1993	Masden et al.
5,283,830	A		2/1994	Hinsley et al. 380/25
5,319,780	A	*	6/1994	Catino et al. 707/8
5,504,883	A		4/1996	Coverston et al.
5,535,375	A	*	7/1996	Eshel et al. 703/27
5,572,711	A		11/1996	Hirsch et al. 395/500
5,596,754	A	*	1/1997	Lomet 709/229
5,604,862	A		2/1997	Midgely et al. 395/182.04
5,617,568	A		4/1997	Ault et al. 395/612
5,628,005	A	*	5/1997	Horvig 707/8
5,649,152	A		7/1997	Ohran et al. 395/441
5,649,196	A		7/1997	Woodhill et al. 395/620
5,668,958	A		9/1997	Bendert et al. 710/128
5,675,726	A		10/1997	Hohenstein et al.
5,675,782	A		10/1997	Montague et al. 395/609
5,689,701	A		11/1997	Ault et al. 395/610
5,720,029	A		2/1998	Kern et al.
5,721,916	A		2/1998	Pardikar 395/617
5,737,523	A		4/1998	Callaghan et al. 395/187.01
5,737,744	A		4/1998	Callison et al.
5,740,367	A	*	4/1998	Spilo 709/201
5,742,752	A		4/1998	DeKoning
5,761,669	A		6/1998	Montague et al. 707/103
5,819,292	A		10/1998	Hitz et al. 707/203
5,819,310	A		10/1998	Vishlitzky 711/114
5,825,877	A		10/1998	Dan et al. 380/4
5,828,876	A	*	10/1998	Fish et al. 707/1
5,835,953	A		11/1998	Ohran 711/162
5,876,278	A		3/1999	Cheng 454/184
5,890,959	A		4/1999	Pettit et al. 454/184
5,915,087	A		6/1999	Hammond et al. 395/187.01
5,931,935	A		8/1999	Calbrera et al. 710/260
5,940,828	A	*	8/1999	Anaya et al. 707/8
5,963,962	A		10/1999	Hitz et al. 707/202
5,996,086	A		11/1999	Delaney et al. 714/4
5,999,943	A		12/1999	Nori et al. 707/104
6,067,541	A		5/2000	Raju et al.
6,085,234	A	*	7/2000	Pitts et al. 709/217
6,275,953	B1	*	8/2001	Vahalia et al. 714/11
6,279,011	B1		8/2001	Muhlestein
2001/0039622	A1		11/2001	Hitz et al.
2002/0019936	A1		2/2002	Hitz et al.

FOREIGN PATENT DOCUMENTS

EP	0308056	A2	3/1989
EP	0410630	A	1/1991
EP	0477039	A2	3/1992
EP	0477039	A3	3/1992
EP	0 477 039	A	3/1992
EP	0537098		4/1993
EP	0566967	A	10/1993
EP	0760503	A1	3/1997
JP	1-273395		11/1989 ..... 361/695
WO	WO 8903086	A1	4/1989
WO	WO 9930254	A1	6/1999
WO	WO 9945456	A1	9/1999
WO	WO 9966401	A1	12/1999
WO	WO 0131446	A1	5/2001
WO	WO 02/29572	A2	4/2002

OTHER PUBLICATIONS

"Migrated Data Backup Utility", IBM Technical Disclosure Bulletin., vol. 37, No. 06B, Jun. 1994, pp. 505–507, XP000456079, IBM Corp. New York., US ISSN: 0018–8689.

R. Reichel: "Inside Windows NT Security:Part 1" Windows/DOS Developers' Journal, vol. 4, No. 4, Apr. 1993, pp. 6–19, XP002107445, Lawrence, Ks, USA.

Borr A J: "SecureShare: safe Unix/Windows file sharing through multiprotocol locking" Proceeding of the 2nd Usenix Windows NT Symposiom, proceedings of 2nd Usenix Windows NT Symposiom, Seattle, WA, USA, Aug., 3–5, 1998, pp. 117–126, XP002097387 ISBN 1–880446–95–2, 1998, Berkeley, CA, USA, Usenix Assoc. USA.

Tanner J:"CIFS: Common Internet File System" Unix Review, vol. 31, Feb. 1997, pp. 31/32, 34, XPOOO783952 see whole document, relevant to claim No. 1–38.

* cited by examiner

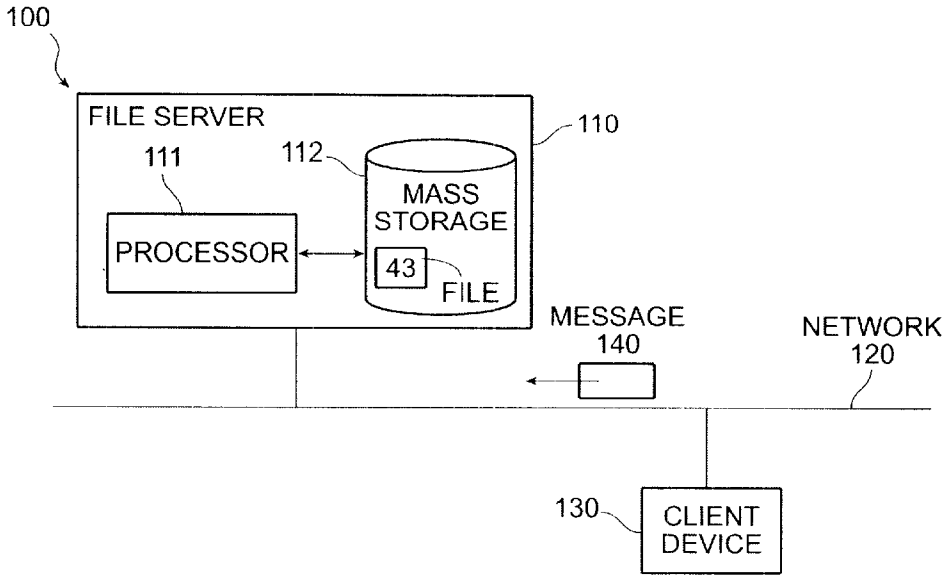

100
FILE SERVER
110
111
112
MASS STORAGE
PROCESSOR
43
FILE
MESSAGE 140
NETWORK 120
130
CLIENT DEVICE

FIG. 1

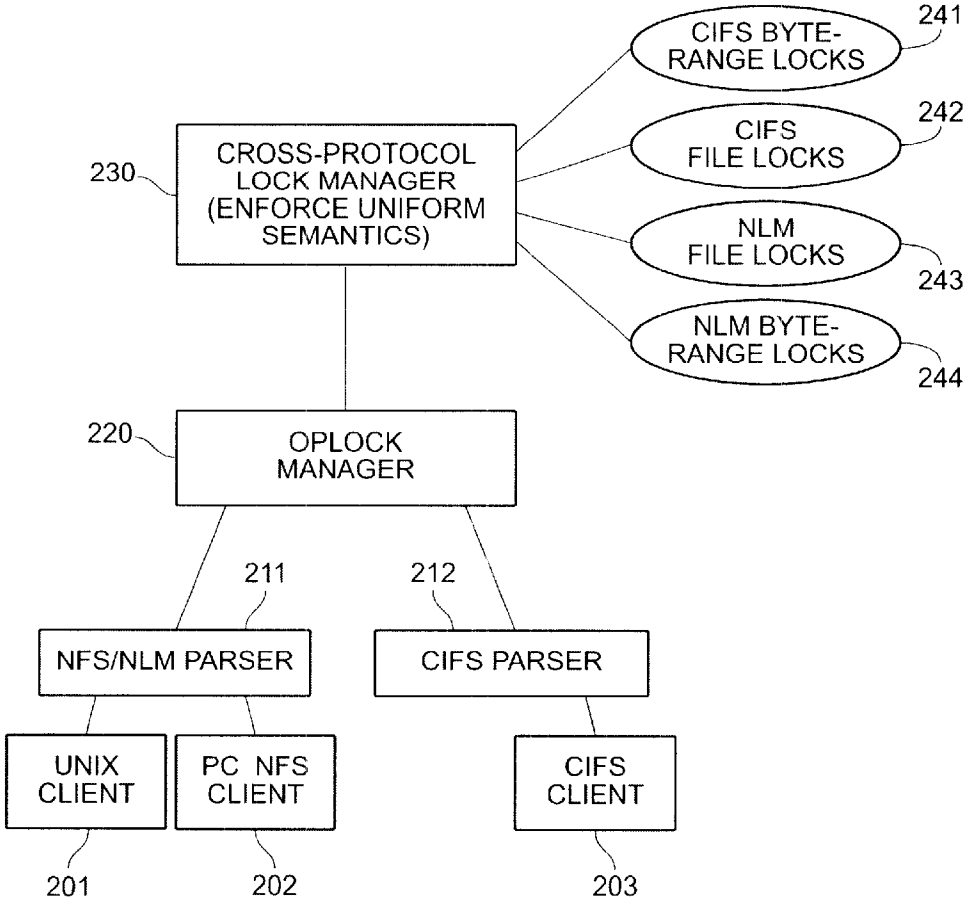

241
CIFS BYTE-RANGE LOCKS
242
CIFS FILE LOCKS
230
CROSS-PROTOCOL LOCK MANAGER (ENFORCE UNIFORM SEMANTICS)
NLM FILE LOCKS
243
NLM BYTE-RANGE LOCKS
244
220
OPLOCK MANAGER
211
212
NFS/NLM PARSER
CIFS PARSER
UNIX CLIENT
PC NFS CLIENT
CIFS CLIENT
201
202
203

FIG. 2

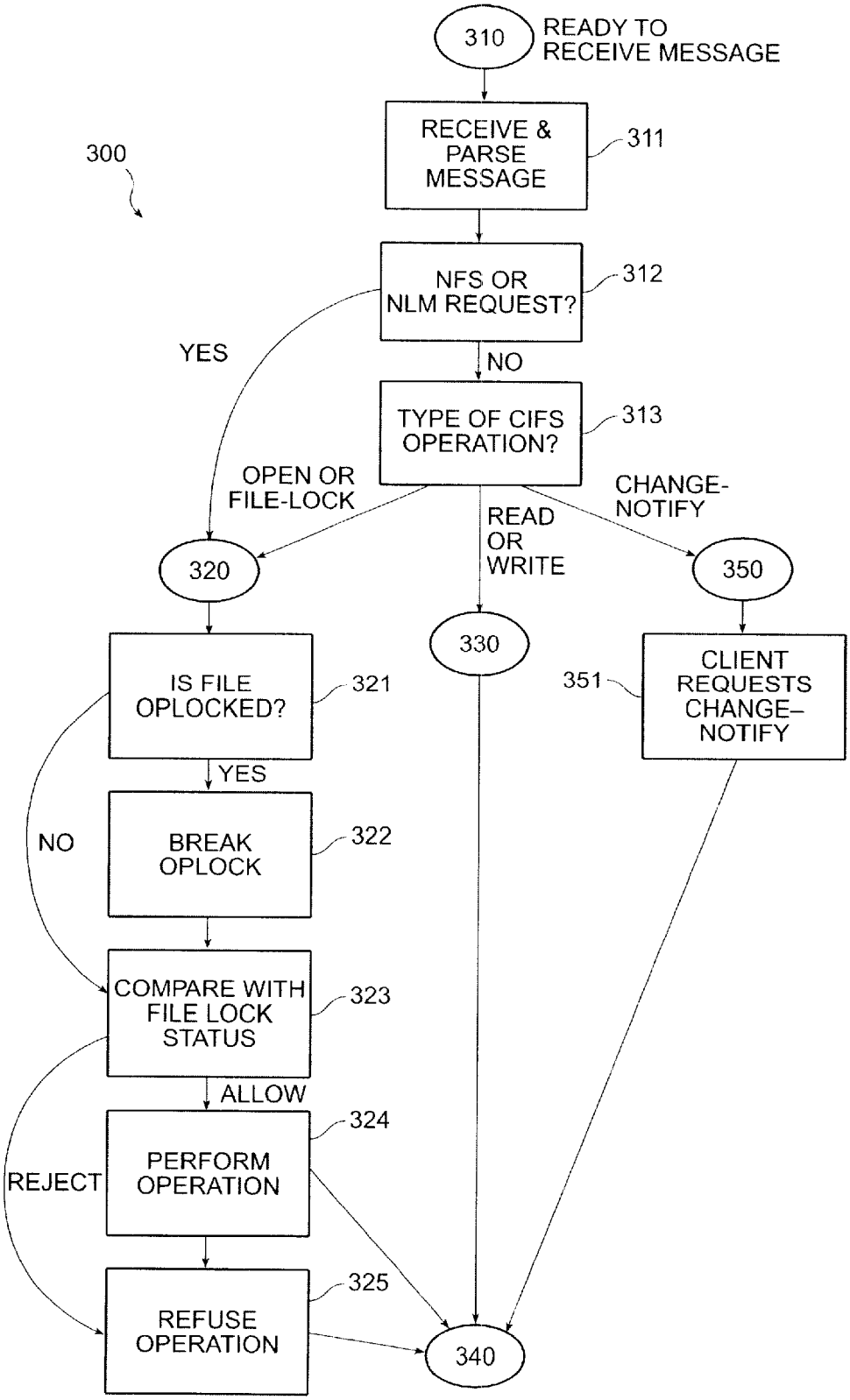

310
READY TO RECEIVE MESSAGE
300
RECEIVE & PARSE MESSAGE
311
NFS OR NLM REQUEST?
312
YES
NO
TYPE OF CIFS OPERATION?
313
OPEN OR FILE-LOCK
READ OR WRITE
CHANGE-NOTIFY
320
330
350
IS FILE OPLOCKED?
321
351
CLIENT REQUESTS CHANGE–NOTIFY
YES
NO
BREAK OPLOCK
322
COMPARE WITH FILE LOCK STATUS
323
ALLOW
REJECT
PERFORM OPERATION
324
REFUSE OPERATION
325
340

FIG. 3

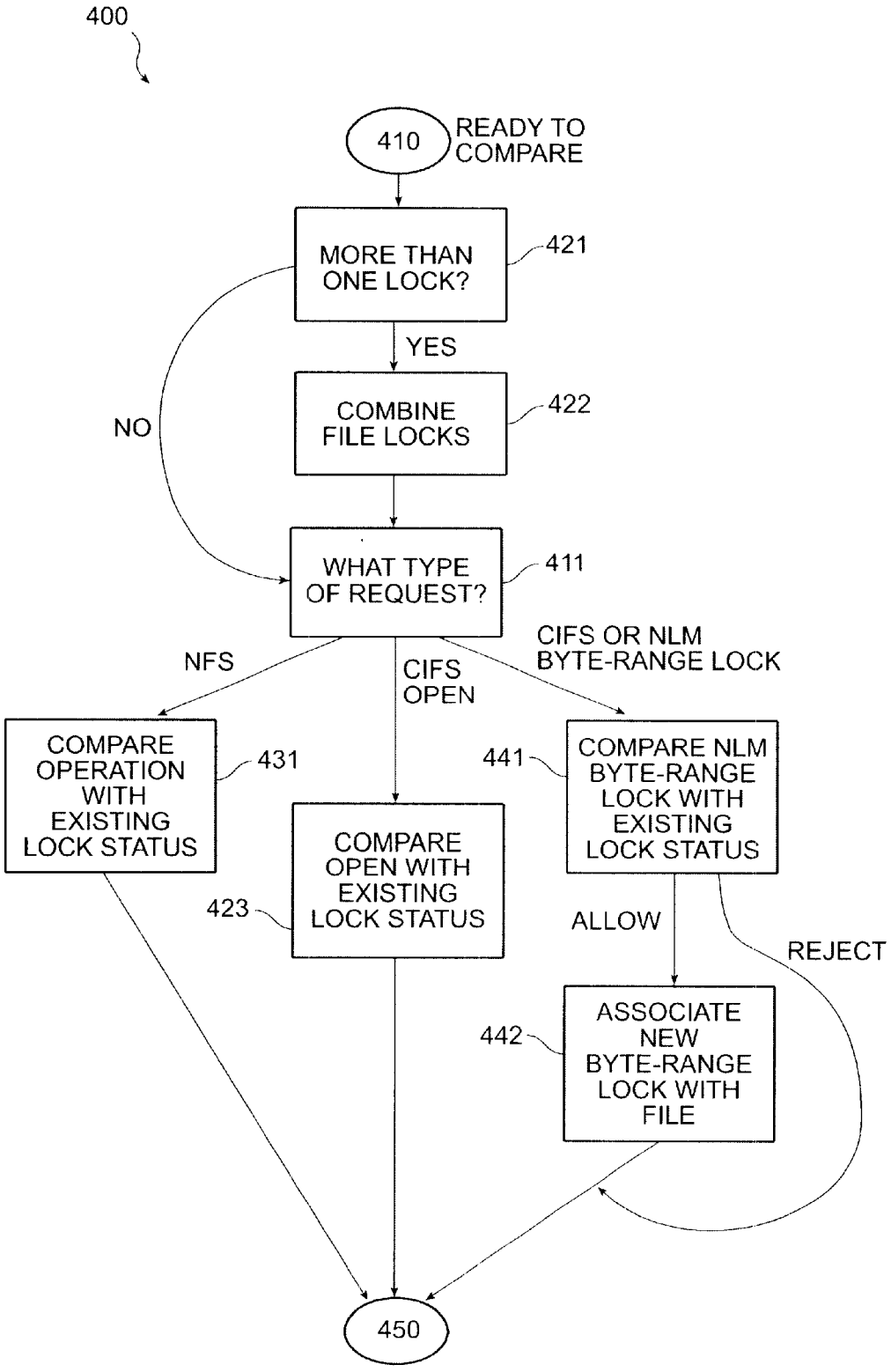

400
410
READY TO COMPARE
MORE THAN ONE LOCK?
421
YES
NO
COMBINE FILE LOCKS
422
WHAT TYPE OF REQUEST?
411
NFS
CIFS OPEN
CIFS OR NLM BYTE-RANGE LOCK
COMPARE OPERATION WITH EXISTING LOCK STATUS
431
COMPARE OPEN WITH EXISTING LOCK STATUS
423
441
COMPARE NLM BYTE-RANGE LOCK WITH EXISTING LOCK STATUS
ALLOW
REJECT
442
ASSOCIATE NEW BYTE-RANGE LOCK WITH FILE
450

FIG. 4

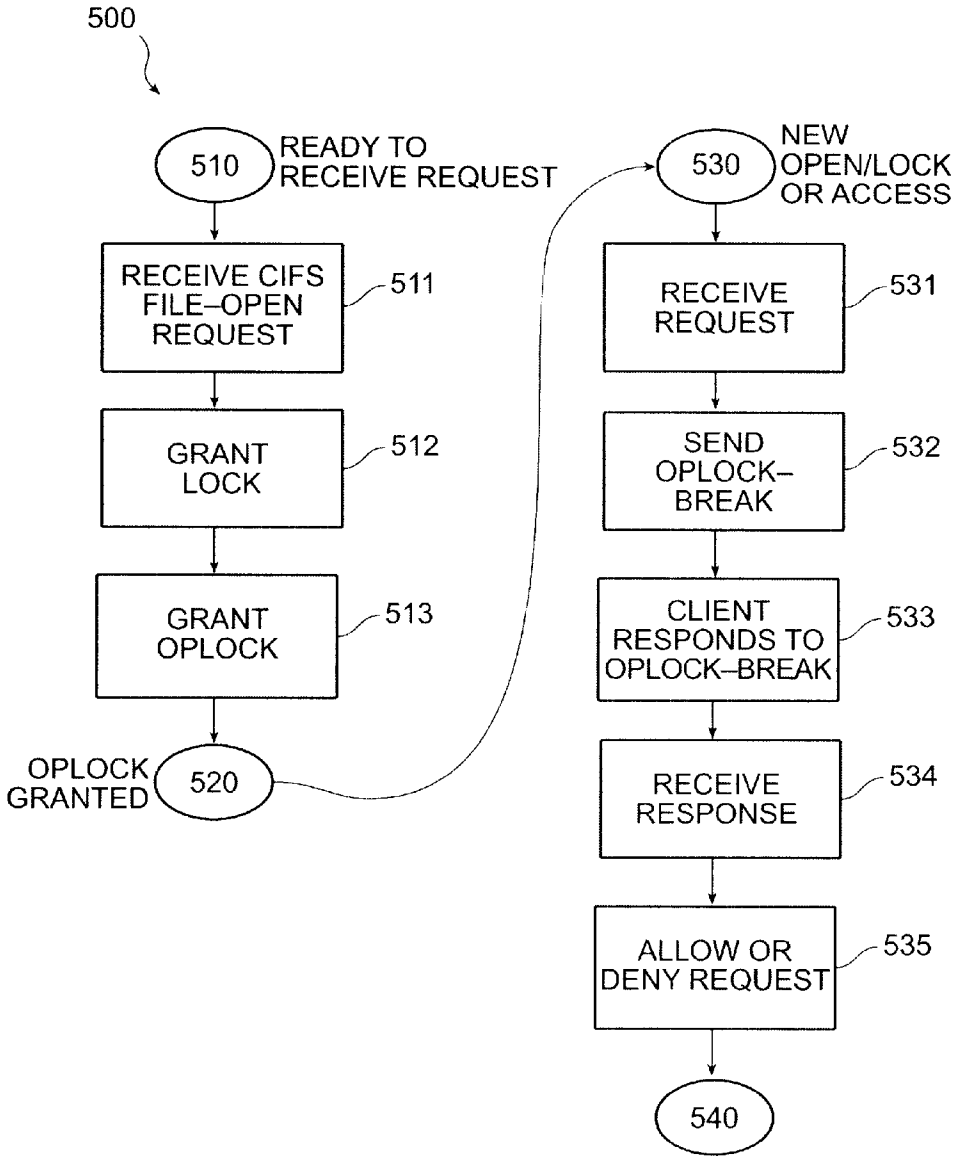

500
510
READY TO RECEIVE REQUEST
RECEIVE CIFS FILE–OPEN REQUEST
511
GRANT LOCK
512
GRANT OPLOCK
513
OPLOCK GRANTED
520
530
NEW OPEN/LOCK OR ACCESS
RECEIVE REQUEST
531
SEND OPLOCK–BREAK
532
CLIENT RESPONDS TO OPLOCK–BREAK
533
RECEIVE RESPONSE
534
ALLOW OR DENY REQUEST
535
540

FIG. 5

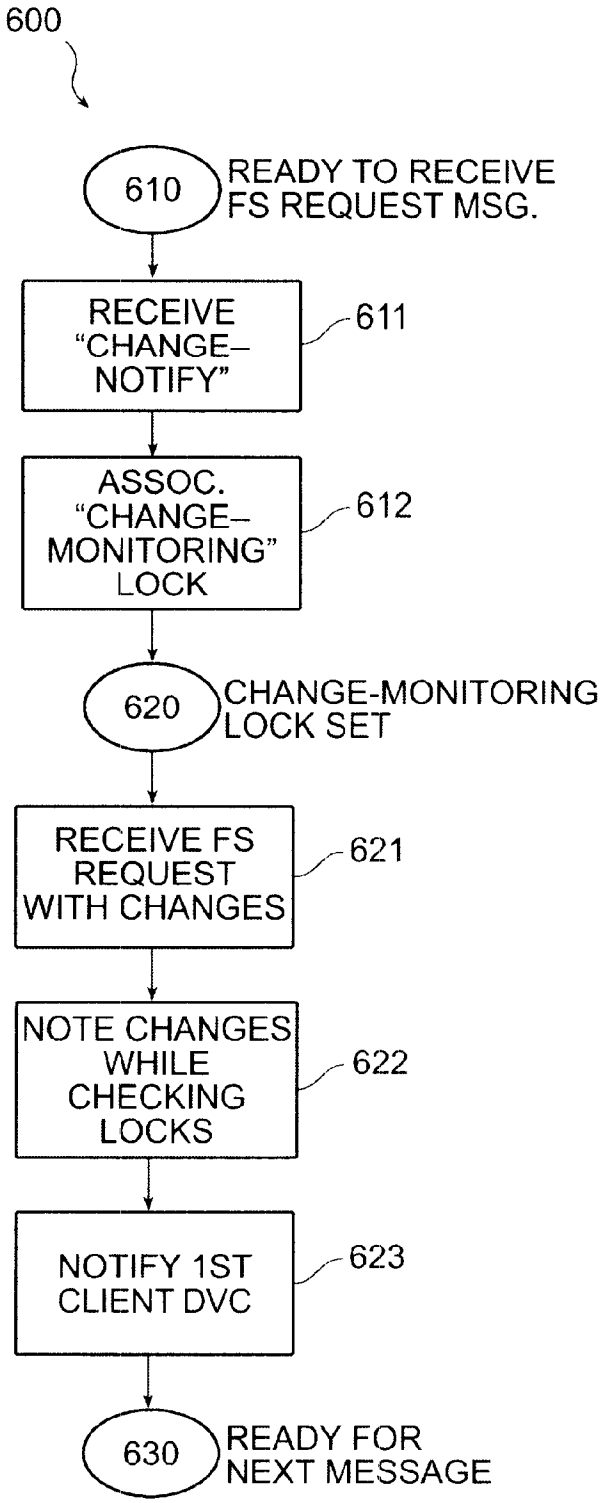

600
610
READY TO RECEIVE FS REQUEST MSG.
RECEIVE "CHANGE–NOTIFY"
611
ASSOC. "CHANGE–MONITORING" LOCK
612
620
CHANGE-MONITORING LOCK SET
RECEIVE FS REQUEST WITH CHANGES
621
NOTE CHANGES WHILE CHECKING LOCKS
622
NOTIFY 1ST CLIENT DVC
623
630
READY FOR NEXT MESSAGE

FIG. 6

ENFORCING UNIFORM FILE-LOCKING FOR DIVERSE FILE-LOCKING PROTOCOLS

This application is a continuation of application Ser. No. 08/985,398, filed Dec. 5, 1997 now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to locking in a multi-protocol file server.

2. Related Art

In an integrated computer network, it is desirable for multiple client devices to share files. One known method is to provide a network file server for storing files, ca pable of receiving and responding to file server requests from those client devices. These file server requests are made using a file server protocol, which is recognized and adhered to by both the file server and the client device. Because the files are stored at the file server, multiple client devices have the opportunity to share simultaneous access to files.

One problem in the art is that there are multiple diverse file server protocols, each with differing semantics for file operations. It is known to provide a single file server that recognizes multiple file server protocols, but there is a particular difficulty in that many file server protocols have differing and incompatible semantics for file locking and file sharing. Incompatible locking semantics presents a hurdle in providing a single file system to multiple diverse client devices. If a first client device relies on a first file server protocol (having first file-locking semantics), a second client device using a second file server protocol (having different file-locking semantics) can cause applications at that first client device to fail catastrophically. Thus, correct operation of each file server protocol relies on conformity with its file-locking semantics by all other file server protocols.

For example, one protocol commonly used for devices using the Unix operating system (or a variant thereof) is the NFS ("Network File System") protocol. Devices using the Windows operating system (or a variant thereof) can also use the NFS protocol by means of the "PC NFS" implementation. The NFS protocol is designed to be stateless, and so does not provide any semantics for files to be locked against sharing or otherwise restricted to a single client device. In contrast, one protocol commonly used for devices using the Windows operating system is the CIFS ("Common Internet File System") protocol. The CIFS protocol has an extensive mandatory file-locking semantics, which CIFS client devices rely on and expect to be adhered to.

In known systems, the NFS protocol has been augmented with an adjunct file-locking protocol, NLM ("Network Lock Manager"), but the NFS protocol treats NLM locks as merely advisory. While this method achieves the purpose of providing file-locking semantics to those NFS applications that use it, it does not prevent NFS applications from ignoring those file-locking semantics, nor does it allow client devices to rely on the file-locking semantics of multiple diverse file server protocols.

Accordingly, it would be desirable to provide a method and system for enforcing file-locking semantics among client devices using multiple diverse file server protocols. This advantage is achieved in an embodiment of the invention in which a uni form set of file-locking semantics is integrated into the kernel of a multi-protocol file server and enforced for client devices using any of the diverse file server protocols recognized by the server. In a preferred

embodiment, specific file-locking semantics of the CIFS protocol are implemented so as to allow NFS client devices to inter-operate with CIFS client devices so as to protect data integrity during client access to a shared file system resident on a network file server.

SUMMARY OF INVENTION

The invention provides a method and system for correct interoperation of multiple diverse file server protocols. A file server recognizing multiple diverse file server protocols provides a uniform multi-protocol lock management system (including a uniform file-locking semantics), which it enforces for all client devices and all recognized file server protocols. In a preferred embodiment, a first file server protocol (such as CIFS) enforces mandatory file-open and file-locking semantics together with an opportunistic file-locking technique, while a second file server protocol (such as NFS, together with an adjunct protocol NLM) lacks file-open semantics and provides only for advisory byte-range and file locking semantics.

The uniform file-locking semantics provides that the file server determines, before allowing any client device to read or write data, or to obtain a new file lock or byte-range lock, whether that would be inconsistent with existing locks, regardless of originating client device and regardless of originating file server protocol or file-locking protocol for those existing locks. In the case of CIFS client devices attempting to read or write data, the file server performs this check when the client device opens the file. In the case of CIFS client devices requesting a byte-range lock, the file server performs this check when the client device requests the byte-range lock. In the case of NFS client devices, the file server performs this check when the client device actually issues the read or write request, or when the NFS client device requests an NLM byte-range lock indicating intent to read or write that byte range. Enforcing file-locking semantics protects file data against corruption by NFS client devices.

In a second aspect of the invention, a CIFS client device, upon opening a file, can obtain an "oplock" (opportunistic lock), an exclusive file lock that permits only that one client to read or write the file. When a client device issues a non-CIFS (that is, NFS or NLM) protocol request for the oplocked file, the file server sends an oplock-break message to the CIFS client device, giving the CIFS client device the opportunity to flush any cached write operations and possibly close the file. Allowing NFS and NLM requests to break oplocks ensures that file data remains available to NFS client devices simultaneously with protecting integrity of that file data.

In a third aspect of the invention, a CIFS client device can obtain a "change-monitoring" lock for a directory in the file system, so as to be notified by the file server whenever there is a change to that directory. (Changes to a directory include creating, deleting or renaming files within that directory, or moving files into or out of that directory.) The file server notes changes to the directory by both CIFS client devices and non-CIFS client devices, and notifies those CIFS client devices with "change-monitoring" locks of those changes.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a first block diagram of a system including a multi-protocol file server.

FIG. **2** shows a second block diagram of a system including a multi-protocol file server.

FIG. **3** shows a process flow diagram of a method of operating a multi-protocol file server.

FIG. **4** shows a process flow diagram of a method of operating a cross-protocol lock manager in a multi-protocol file server.

FIG. **5** shows a process flow diagram of a method of operating an oplock manager in a multi-protocol file server.

FIG. **6** shows a process flow diagram of a method of operating a change-notify manager in a multi-protocol file server.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following description, a preferred embodiment of the invention is de scribed with regard to preferred process steps and data structures. Those skilled in the art would recognize after perusal of this application that embodiments of the invention can be implemented using general purpose processors or special purpose processors or other circuits adapted to particular process steps and data structures described herein, and that implementation of the process steps and data structures described herein would not require undue experimentation or further invention.

System Architecture (Client/Server)

FIG. **1** shows a first block diagram of a system including a multi-protocol file server.

A system **100** includes a file server **110**, a computer network **120**, and a plurality of client devices **130**.

The file server **110** includes a processor **111** and mass storage **112**. The mass storage **112** is capable of storing and retrieving a set of files **113** having data for storage and retrieval. The processor **111** is capable of receiving a sequence of request messages **140** from the network **120**, parsing those messages as commands and data, and manipulating the files **113** on the mass storage **112**, and sending response messages, in response to those commands and data.

The file server **110** and the client devices **130** are coupled to the network **120** and communicate using messages **140** transmitted on the network **120**. The messages **140** include file system requests transmitted by the client devices **130** to the file server **110** and file system responses transmitted by the file server **110** to the client devices **130**.

System Architecture (File-Locking Semantics)

FIG. **2** shows a second block diagram of a system including a multi-protocol file server.

The system **100** includes the set of client devices **130**, including Unix client devices **201**, PC NFS Windows client devices **202**, and CIFS Windows client devices **203**. Unix client devices **201** execute the Unix operating system and use the Unix/NFS file server protocol. PC NFS Windows client devices **202** execute the Windows operating system and use the PC NFS file server protocol. CIFS Windows client devices **203** execute the Windows operating system and use the CIFS file server protocol.

Unix client devices **201** and PC NFS Windows client devices **202** communicate with the file server **110** using the NFS file server protocol, which is recognized at the file server **110** by an NFS file server protocol parser **211**. CIFS Windows client devices **203** communicate with the file server **110** using the CIFS file server protocol, which is recognized at the file server **110** by a CIFS file server protocol parser **212**. Messages using either the NFS file server protocol or the CIFS file server protocol are parsed by the processor **111** and processed by an oplock manager **220**.

The oplock manager **220** manages access to files **113** having CIFS oplocks. Its operation is described in further detail with regard to FIG. **3** and FIG. **5**. The oplock manager element **220** is coupled to a cross-protocol lock manager **230**.

The cross-protocol lock manager **230** manages the file-locking semantics of the file server **110**. It processes and records information regarding four types of locks—CIFS byte-range locks **241**, CIFS file locks **242**, PC NFS (NLM) file locks **243**, and NLM byte-range locks **244**. The operation of the cross-protocol lock manager **230** in enforcing uniform file-locking semantics is described in further detail with regard to FIG. **3** and FIG. **4**.

Differing File-Locking Semantics

As noted with regard to FIG. **2**, file server request messages **140** can be received from Unix client devices **201**, PC NFS Windows client devices **202**, or CIFS Windows client devices **203**, and can use the NFS file server protocol or the CIFS file server protocol. In addition to each using differing file server protocols, each of these types of client device **130** has a different model of file locking provided by the file server **110**.

In particular, the NFS file server protocol provides for performing file system operations without any form of file-open or file-close semantics. These NFS file system operations can include read or write operations to file data, or file system manipulation (that is, read and write operations to directories). File system manipulation can include creating files or directories, renaming files or directories, moving files from one directory to another, or removing (deleting) files or directories from the file system.

The NLM adjunct protocol provides for obtaining and releasing byte-range locks for files. These byte-range locks can be "read locks," which induce other compliant applications (such as at other client devices **130**) to refrain from writing to the specified byte-range. These byte-range locks can alternatively be "write locks," which induce other compliant applications to refrain from either reading from or writing to the specified byte-range.

The CIFS file server protocol provides for performing file-open operations, and obtaining file locks on the files **113** to be opened, before attempting any read or write operations to data in those files **113**. At file-open time, a CIFS client device **130** can specify the access-mode it desires (read-only, write-only, or read-write), and the deny-mode desires to impose on other client devices **130** attempting to open the same file **113** (deny-none, deny-read, deny-write, or deny-all). Thereafter, CIFS file system operations need only be checked against the access-mode that the file-open obtained. A CIFS client device **130** can also specify a byte-range lock for a byte-range in a file the client device **130** holds open. The byte-range lock is either an exclusive lock, also called a "write lock" (having access-mode read-write and deny-mode deny-all), or a nonexclusive lock, also called a "read lock" (having access-mode read-only and deny-mode deny-write).

The file server **110** determines a lock mode that combines the access-mode and the deny-mode. As used herein, the phrase "lock mode" refers to a uniform lock mode imposed by the file server **110** which combines an access-mode and a deny-mode.

At file-open time, CIFS client devices **130** can also obtain an oplock (opportunistic lock), which provides that the CIFS client device **130** opening the file has exclusive access to the file so long as another client device **130** does not attempt to use the file. The oplock provides a higher degree of exclusivity to the file than strictly necessary for the client device **130**, with the caveat that the exclusivity of the oplock can be broken by attempted access by another client device **130**.

The file server **110** provides for correct interoperation among client devices **130** using NFS (with or without the adjunct protocol NLM) or CIFS. To provide for correct

interoperation, the file server **110** provides a uniform file-locking semantics. In a preferred embodiment, the uniform file-locking semantics has the following effects:

- The file server **110** prevents Unix client devices **201** from performing NFS write operations that would overwrite data in a file **113** that is already opened and in use by a CIFS client with deny-modes deny-write or deny-all.
- The file server **110** prevents Unix client devices **201** and PC NFS Windows client devices **202** from performing NFS file system operations that would remove or re-name a file **113** that is already opened and in use by a CIFS client.
- When a Unix client device **201** or a PC NFS Windows client device **202** makes an NFS request to remove, rename, or write data to a file **113** that is oplocked by a CIFS client, the file server **110** will enforce CIFS oplock semantics for the file **113**. The file server **110** sends an oplock-break message **140** to the client device **130** holding the oplock, and receives a response from the client device **130**. If the client device **130** closes the file **113**, the NFS request can proceed and the file server **110** allows it to.
- When a Unix client device **201** or a PC NFS Windows client device **202** makes an NFS request to read data from a file **113** that is oplocked by a CIFS client, the file server **110** will enforce CIFS oplock semantics for the file **113**. The file server **110** sends an oplock-break message **140** to the client device **130** holding the oplock, and receives a response from the client device **130**. When the client device **130** either closes the file **113** or flushes its write cache to the file server **110**, the NFS request can proceed and the file server **110** allows it to.
- The file server **110** tests for mutual compatibility for file-open requests from CIFS Windows client devices **203**, and NLM file lock requests from PC NFS Windows client devices **202**, with regard to their specified lock modes. The phrase NLM "file lock" is used herein in place of the known phrase NLM "share lock," further described in the following document: "X/Open CAE Specification: Protocols for X/Open Interworking: XNFS, Issue 4 (X/Open Document Number C218), hereby incorporated by reference as if fully set forth herein. The specified lock mode is determined by the file server **110** by combining the requested access-mode and deny-mode.

To provide these effects, the file server **110** performs the following lock management operations:

- Upon receiving a CIFS file-open request message **140**, the file server **110** tests the file-open request for conflict with existing CIFS and NLM file locks, and for conflict with existing NLM byte-range locks. For the purpose of comparison with newly requested file locks, the file server **110** treats existing NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks and access-mode read-write for exclusive locks.
- Upon receiving a CIFS byte-range lock request message **140**, the file server **110** tests the byte-range lock request for conflict with existing CIFS and NLM byte-range locks.
- Upon receiving an NLM byte-range lock request message **140**, the file server **110** tests the byte-range lock request for conflict with existing CIFS and NLM byte-range locks, and for conflict with existing CIFS file locks.

Upon receiving an NLM file lock request message **140** from a PC NFS client device **130** (used to simulate a file-open request message **140**), the file server **110** tests the NLM file lock request for conflict with existing CIFS and NLM file locks, and for conflict with existing NLM byte-range locks. For the purpose of comparison with newly requested NLM file locks, the file server **110** treats existing NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks and access-mode read-write for exclusive locks.

Method of Operation (Multi-Protocol File Server)

FIG. **3** shows a process flow diagram of a method of operating a multi-protocol file server.

A method **300** of operating a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **310**, the file server **110** is ready to receive the file server request message **140**.

At a step **311**, the file server **110** receives and parses the file server request message **140**. The file server **110** determines if the file server request message **140** uses the NFS file server protocol, the NLM file locking protocol, or the CIFS file server protocol. If the file server request message **140** uses the NFS file server protocol or the NLM file locking protocol, the method **300** continues with the step **312**. If the file server request message **140** uses the CIFS file server protocol, the method **300** continues with the step **313**.

At a step **312**, the file server **110** determines if the request message **140** includes an NFS file server request to perform a file system operation (such as to read or write data, or to modify a directory). Alternatively, the file server **110** determines if the request message **140** includes an NLM file-locking request to obtain an NLM byte-range lock. In either case, the method **300** continues at the flow point **320**.

At a step **313**, the file server **110** determines if the file server request message **140** is to perform a CIFS read or write operation, to obtain a CIFS byte-range lock, or to perform a CIFS file-open operation. In the file server request message **140** is to obtain a CIFS byte-range lock or to perform a CIFS file-open operation, the method **300** continues at the flow-point **320**. If the file server request message **140** is to perform a CIFS read or write operation, the method continues at the flow-point **330**. If the file server request message **140** is a CIFS "change-notify" request, the method continues at the flow point **350** (the change-notify request is further described with regard to FIG. **6**).

At a flow point **320**, the file server **110** is ready to compare the operation requested by the file server request message **140** with the file-locking status of the file **113**. The file-locking status of the file **113** includes existing file locks and byte-range locks for the file **113**.

At a step **321**, the file server **110** determines the file **113** that is the subject of the file server request message **140**, and determines if the file **113** is oplocked. If the file **113** is oplocked, the method **300** continues with the step **322**. If the file **113** is not oplocked, the method **300** continues with the step **323**.

At a step **322**, the file server **110** breaks the oplock, as described herein. Performance of the step **322** is further described with regard to FIG. **5**. Breaking the oplock can cause the file-locking status of the file **113** to change.

At a step **323**, the file server **110** compares the requested operation with the file-locking status of the file **113**, using a uniform file-locking semantics. In this step, the requested operation can be an NFS read or write operation, an NFS or CIFS directory modification operation, an attempt to obtain an NLM file lock or byte-range lock, or a CIFS file-open operation. Performance of the step **323** and the uniform

file-locking semantics are further described with regard to FIG. **4**. If the comparison shows that the requested operation is allowable, the method **300** continues with the step **324**. If the requested operation is not allowable, the method **300** continues with the step **325**.

At a step **324**, the file server **110** performs the requested operation. The method **300** continues at the flow point **340**.

At a step **325**, the file server **110** refuses to perform the requested operation and responds to the client device **130** with an error message. The method **300** continues at the flow point **340**.

At a flow point **330**, the file server **110** is ready to compare the operation requested by the file server request message **140** with the file-locking status of the file **113**.

At a flow point **350**, the file server **110** is ready to perform the change-notify operation, as described herein.

At a step **351**, a first CIFS client device **130** requests a file lock for a directory (using a file system request message **140** to open the directory), and converts the file lock for the directory to a change-monitoring lock on the directory. Performance of this step **351** is further described with regard to FIG. **6**.

At a flow point **340**, the file server **110** has responded to the file server request message **140**, and the method **300** is complete with regard to that file server request message **140**.

Method of Operation (Cross-Protocol Lock Manager)

FIG. **4** shows a process flow diagram of a method of operating a cross-protocol lock manager in a multi-protocol file server.

A method **400** of operating a cross-protocol lock manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **410**, the file server **110** is ready to compare the requested operation in the file server request message **140**, with the file-locking status of the file **113**.

The file server **110** uses a uniform file-locking semantics, so as to model file-locking aspects of any requested operation from any file server protocol in the same way. The uniform file-locking semantics identifies a uniform set of file locks, each including an access-mode for the requesting client device **130** and a deny-mode for all other client devices **130**.

In a preferred embodiment, the access-mode can be one of three possibilities—read-only, write-only, or read-write. Similarly, in a preferred embodiment, the deny-mode can be one of four possibilities—deny-none, deny-read, deny-write, or deny-all.

After a first client device **130** obtains a file lock for a file **113**, a second client device **130** can only access that file **113** if the lock mode determined by the file server **110** to be requested by the second client device **130** is compatible with the file-locking status of the file **113**. For example, a first client device **130** can obtain a file lock for a file **113** with a deny-mode deny-write. A second NFS client device **130** could attempt to write to the file **113**, or a second CIFS client device **130** could attempt to open the file **113** with an access-mode including write access. In either such case (if the file lock for the file **113** is not an opportunistic lock, as further described herein), the file server **110** will deny the request by the second client device **130**.

As noted herein, the file server **110** performs the comparison of the file lock with the access requested by the second client device **130** at differing times, in response to the file server protocol used by the second client device **130**:

- If the second client device **130** uses the CIFS file server protocol to open the file **113**, the file server **110** checks the file-locking status of the file **113** at file-open time.
- If the second client device **130** uses the NFS file server protocol to read or write to the file **113**, the file server **110** checks the file-locking status of the file **113** at the time of the actual file system operation. This also applies to file system operations that have the effect of removing the file from view of the first client device **130**, such as operations to move, remove, or rename the file **113**.
- If the second client device **130** uses the CIFS file server protocol to request a byte-range lock, the file server **110** checks the file-locking status of the file **113** for conflict with other CIFS or NLM byte-range locks, at the time the byte-range lock is requested. The file server **110** does not check for conflict with other CIFS file locks at the time the byte-range lock is requested, because those were checked at file-open time.
- If the second client device **130** uses the NLM protocol to request a byte-range lock, the file server **110** checks the file-locking status of the file **113** for conflict with existing CIFS or NLM byte-range locks, and for conflict with existing CIFS file locks, at the time the byte-range lock is requested.

At a step **421**, the file server **110** determines if there is already more than one file lock associated with the file **113**. If so, the method **400** continues with the step **422**. If not, the method continues with the step **411**.

At a step **422**, the file server **110** combines file locks already associated with the file **113** into a single equivalent file lock associated with the file **113**. To perform this step **422**, the file server **110** cross-indexes in table 1 a cumulative file lock with each pre-existing file lock, until all pre-existing file locks have been cumulated together.

Table 1 shows a lock conversion table in a multi-protocol file server with unified file-locking semantics.

TABLE 1

Lock Conversion Matrix

	Existing file lock mode									
	NULL	A: R D: DN	A: W D: DN	A: RW D: DN	A: R D: DR	A: W D: DR	A: RW D: DR	A: R D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
New lock mode											
NULL	NULL	A: R D: DN	A:W D: DN	A: RW D: DN	A: R D: DR	A:W D: DR	A: RW D: DR	A: R D: DW	A:W D: DW	A: RW D: DW	A: Any D: DA

TABLE 1-continued

Lock Conversion Matrix

	Existing file lock mode										
	NULL	A: R D: DN	A: W D: DN	A: RW D: DN	A: R D: DR	A: W D: DR	A: RW D: DR	A: R D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
A: R D: DN	A: R D: DN	A: R D: DN	A: RW D: DN	A: RW D: DN	A: R D: DR	A: RW D: DR	A: RW D: DR	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
A: W D: DN	A: W D: DN	A: RW D: DN	A: W D: DN	A: RW D: DN	A: RW D: DR	A: W D: DR	A: RW D: DR	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
A: R D: DR	A: R D: DR	A: R D: DR	A: RW D: DR	A: RW D: DR	A: R D: DR	A: RW D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
A: W D: DR	A: W D: DR	A: RW D: DR	A: W D: DR	A: RW D: DR	A: RW D: DR	A: W D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
A: R D: DW	A: R D: DW	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
A: W D: DW	A: W D: DW	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
A: RW D: DW	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA

A = Access Mode (R = Read, W = Write, RW = Read-Write, Any = any one of R or W or RW)
D = Deny Mode (DN = Deny None, DR = Deny Read, DW = Deny Write, DA = Deny All)

At a step **411**, the file server **110** determines the nature of the requested operation in the file server request message **140**. If the requested operation is a CIFS file-open operation, the method **400** continues with the step **423**. If the requested operation is an NFS file server operation, the method **400** continues with the step **431**. If the requested operation is either a CIFS request an NLM request for a byte-range lock, the file system **110** continues with the step **441**.

At a step **423**, the file server **110** compares the file lock already associated with the file **113** with the file open requested by the second client device **130**. To perform this step **423**, the file server **110** cross-indexes in table 2 the pre-existing file lock and the requested new access-mode and deny-mode, and allows or denies the requested new access-mode and deny-mode in response to the associated table entry.

If the file server **110** allows the requested new access-mode and deny-mode, the method **400** performs the step **424**. If the file server **110** denies the requested new access-mode and deny-mode, the method **400** does not perform the step **424**.

Table 2 shows a cross-index of attempted file locks in a multi-protocol file server with unified file-locking semantics.

TABLE 2

Multi-Protocol Lock Compatibility Matrix

	Pre-existing file lock										
	NULL	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
New mode being requested											
NULL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A: R D: DN	✓	✓	X	✓	✓	X	✓	✓	X	✓	X
A: R D: DR	✓	X	X	X	✓	X	✓	✓	X	✓	X
A: R D: DW	✓	✓	X	✓	X	X	X	X	X	X	X
A: W D: DN	✓	✓	✓	X	✓	✓	X	✓	✓	X	X
A: W D: DR	✓	X	X	X	✓	✓	X	X	X	X	X
A: W D: DW	✓	✓	✓	X	X	X	X	X	X	X	X
A: RW D: DN	✓	✓	X	X	✓	X	X	✓	X	X	X

TABLE 2-continued

Multi-Protocol Lock Compatibility Matrix

	Pre-existing file lock										
	NULL	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
A: RW D: DR	✓	X	X	X	✓	X	X	X	X	X	X
A: RW D: DW	✓	✓	X	X	X	X	X	X	X	X	X
A: Any D: DA	✓	X	X	X	X	X	X	X	X	X	X

A = Access Mode, D = Deny Mode
✓ = New request will be granted. X = New request will be denied.

As shown in table 2, each pair of pre-existing file lock and requested new access-mode and deny-mode has an associated decision to allow or to deny the requested new access-mode and deny-mode.

If the file server **110** is checking for conflicts between an existing CIFS file lock and a new request to perform a file-open operation, the existing CIFS file lock is cross-indexed against the access-mode and deny-mode requested in the new file-open request.

If the file server **110** is checking for conflicts between existing file locks and a new NFS request to perform a file read or write operation, the aggregate lock mode (the combination of existing file locks) is cross-indexed against the access-mode required to perform the new request.

If the file server **110** is checking for conflicts between existing file locks or byte-range locks, and a new request for a NLM byte-range lock, the existing file locks and byte-range locks are cross-indexed against a lock mode equivalent to the new NLM byte-range lock request. For the purpose of comparing with existing file locks, the file server **110** treats newly requested NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks (also called "read locks") and access-mode read-write for exclusive locks (also called "write locks"). For the purpose of comparing with existing byte-range locks, the file server **110** treats newly requested NLM byte-range locks as having access-mode read-only and deny-mode deny-write for read locks, and as having access-mode read-write and deny-mode deny-all for write locks.

The method **400** then continues at the flow point **450**.

At a step **431**, the file server **110** compares the file-locking status of the file **113** with the operation requested by the second client device **130**. To perform this step **431**, the file server **110** compares the deny-mode for the file lock with the requested operation, and allows or denies the requested operation in response thereto.

The method **400** then continues at the flow point **450**.

At a step **441**, the file server **110** compares the file-locking status of the file **113** with the NLM byte-range lock requested by the second client device **130**. In a preferred embodiment, CIFS byte-range lock requests are only checked against byte-range locks because they require a prior CIFS file open operation at which existing file locks were already checked. To perform this step **441**, the file server **110** cross-indexes in table 3 the pre-existing file-locking status and the requested byte-range lock, and allows or denies the requested byte-range lock in response to the associated table entry.

If the file server **110** allows the requested new NLM byte-range lock, the method **400** performs the step **442**. If the file server **110** denies the requested new byte-range lock, the method **400** does not perform the step **442**.

Table 3 shows a cross-index of existing file locks and newly requested NLM byte-range locks in a multi-protocol file server with unified file-locking semantics.

TABLE 3

Compatibility of new NLM byte-range locks with existing file locks

	Existing lock mode										
	NULL	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
Write Lock	✓	✓	✓	X	✓	✓	X	✓	✓	X	X
Read Lock	✓	✓	X	✓	✓	X	✓	✓	X	✓	X

A =Access Mode, D = Deny mode
✓ = New NLM byte-range lock request will be granted.
X = New NLM byte-range lock request will be denied.

As shown in table 3, each pair of existing file lock and newly requested NLM byte-range lock has an associated decision to allow or to deny the requested new byte-range lock.

At a step **442**, the file server **110** associates the requested new byte-range lock with the file **113** as a new additional byte-range lock.

The method **400** then continues at the flow point **450**.

At a flow point **450**, the file server **110** has compared the requested operation in the file server request message **140** with the file-locking status of the file **113**, and allowed or denied the requested operation.

Method of Operation (Oplock Manager)

FIG. **5** shows a process flow diagram of a method of operating an oplock manager in a multi-protocol file server.

A method **500** of operating a oplock manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

Oplocks are known in the art of file-locking in Windows operating system environments. They are further described in documentation available for the "Windows NT 4.0" operating system, available from Microsoft Corporation of Redmond, Wash., including for example the CIFS IETF specification, available via the FTP protocol at host ftp.microsoft.com, directory /developr/drg/CIFS, files cifs6.doc or cifs6.txt, hereby incorporated by reference as if fully set forth herein.

At a flow point **510**, the file server **110** is ready to receive a request from a CIFS first client device **130** to open a file **113**.

At a step **511**, the file server **110** receives a file-open request for a file **113** from a CIFS first client device **130**. The file-open request designates an access-mode and a deny-mode.

At a step **512**, the file server **110** determines that it should allow the request, and grants the first client device **130** a file lock with the designated access-mode and deny-mode.

At a step **513**, if the client device **130** has requested an oplock on the file open request, the file server **110** grants the first client device **130** an oplock at a level of exclusivity possibly greater than the first client device **130** actually requires.

For example, when a CIFS first client device **130** opens a file **113** with the access-mode read-only and deny-mode deny-write, the file server **110** associates a file lock of that type with the file **113**. The file server **110** further associates an oplock with the file **113** with the access-mode read-write and deny-mode deny-all.

At a flow point **520**, the file server **110** has responded to the request from the CIFS first client device **130** for a file lock for a file **113**.

At a flow point **530**, a second client device **130** attempts to open the file **113**.

At a step **531**, the file server **110** receives either a file-open request from a second CIFS client device **130** or a NLM file lock request from a PC NFS client device **130**.

As part of performing this step **531**, the file server **110** suspends execution of the request by the second client device **130** while it breaks the oplock and obtains a response from the holder of the oplock, the first client device **130**.

At a step **532**, the file server **110** breaks the oplock by sending an "oplock break" message **140** to the CIFS first client device **130**.

When the second client device **130** i s a CIFS client device **130**, this is already expected. When the second client device **130** is an NFS client device **130**, the file server **110** delays its response to the NFS (or NLM) protocol request message **140** until the CIFS first client device **130** responds to the "oplock-break" message **140**.

At a step **533**, the CIFS first client device **130** receives the "oplock-break" message **140**, and can respond to the message **140** in one of two ways:

- The CIFS first client device **130** can close the file **113** (thus removing the file lock associated with the file-open); or
- The CIFS first client device **130** can flush all outstanding CIFS write and byte-range lock requests for the file **113** that are being cached locally at the client device **130** (that is, it can forward the results of those file system operations to the file server **110**), and discard any read-ahead data it has obtained for the file **113**. Read-ahead data should be discarded because the second client device **130** might subsequently write new data to the file, invalidating the read-ahead data.

At a step **534**, the file server **110** receives the response from the CIFS first client device **130**.

At a step **535**, the file server **110** determines if the CIFS first client device **130** has maintained the file **113** open, and if so, compares the lock mode implied by the request by the second client device **130** against the new file-locking status of the file **113**. If the file server **110** determines that the request by the second client device **130** is allowed to proceed, it continues with the flow point **540**. If the file server **110** determines that the request by the second client device **130** is not allowed to proceed, it denies the request.

At a flow point **540**, the file server **110** is ready to proceed to allow the request from the second client device **130** noted in the step **531**.

Method of Operation (Change-Notify Manager)

FIG. **6** shows a process flow diagram of a method of operating a change-notify manager in a multi-protocol file server.

A method **600** of operating a change-notify manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **610**, the file server **110** is ready to receive the file server request message **140**.

At a step **611**, the file server **110** receives a file-open request message **140** from a first CIFS client device **130**, designating a directory on the file server **110**. The file server **110** determines that it should allow the file-open request and grants a CIFS file lock on the directory to the first CIFS client device **130**.

At a step **612**, the file server **110** receives a change-notify request message from the first CIFS client device **130**, referencing the open directory, to convert the file lock on the open directory to a change-monitoring lock.

At a step **613**, the file server **110** converts the file lock on the open directory to a change-monitoring lock on the designated directory.

At a flow point **620**, the "change-monitoring" lock has been associated with the designated directory, and the first CIFS client device **130** is ready to be notified of changes to that directory.

At a step **621**, the file server **110** receives a file server request message **140** from a second client device **130**, requesting a change to the designated directory, and thus triggering a change notification to the first client device **130**. (Types of change include file creation, file deletion, file rename, file move between directories, file attribute change, and file modification time change.) The file server request message **140** from the second client device **130** can be either CIFS or NFS. The second client device **130** can be any one of a Unix NFS client device **201**, a PC NFS client device **202**, or a CIFS Windows client device **203**.

At a step **622**, the file server **110** notifies the first client device **130**, which holds the "change-monitoring" lock, of the changes noted in the step **621**, containing possibly multiple entries, each of which specifies both the name of the changed file **113** or subdirectory within the monitored directory and the type of change. If there is more than one such first client device **130**, the file server **110** notifies all of them.

Change-notification is known in the art of file-locking in Windows NT operating system environments. It is further

described in documentation available for the "Windows NT 4.0" operating system, available from Microsoft Corporation of Redmond, Wash., including for example the CIFS IETF specification, available via the FTP protocol at host ftp.microsoft.com, directory /developr/drg/CIFS, files cifs**6**.doc or cifs**6**.txt, hereby incorporated by reference as if fully set forth herein.

At a flow point **630**, the file server **110** has notified the first CIFS client device **130** of changes to the designated directory, and is ready for a next message **140**.

ALTERNATIVE EMBODIMENTS

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

TECHNICAL APPENDIX

Other and further information about the invention is included in a technical appendix enclosed with this application. This technical appendix includes 30 pages (including drawings) and is hereby incorporated by reference as if fully set forth herein.

What is claimed is:

1. A method of operating a file server, said method including steps for enforcing a uniform file-locking semantics among a set of client devices using a plurality of diverse file-locking protocols, said file server implementing said plurality of diverse file-locking protocols and enforcing said uniform file-locking semantics for said plurality of diverse file locking protocols;

wherein said uniform file-locking semantics includes steps for

granting an opportunistic lock on a selected file to a first said client device in response to a first message using a first protocol; and

breaking said opportunistic lock in response to a second message using a second protocol;

wherein the first protocol and the second protocol are ones of said plurality of diverse file locking protocols.

2. A method as in claim **1**, wherein said uniform file-locking semantics includes said opportunistic lock capable of being requested by said first client device using said first protocol; and

breaking of said opportunistic lock being triggered by said second client device using said second protocol different from said first protocol.

3. A method as in claim **2**, wherein said first protocol includes CIFS.

4. A method as in claim **2**, wherein said second protocol includes NFS or NLM.

5. A method as in claim **1**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

6. A method as in claim **1**, wherein said uniform file-locking semantics includes a change-monitoring lock type capable of being requested by said first client device using said first protocol; and

a change notification being triggered by said second client device using said second protocol different from said first protocol.

7. A method as in claim **6**, wherein said first protocol includes CIFS.

8. A method as in claim **6**, wherein said second protocol includes NFS.

9. A method as in claim **1**, wherein said uniform file-locking semantics includes steps for

granting a change-monitoring lock on a selected directory to said first client device in response to said first message using said first protocol; and

sending a change-notify message to said first client device in response to said second message regarding said selected directory using said second protocol.

10. A method as in claim **1**, wherein said steps for enforcing include steps for

recognizing said plurality of diverse file locking protocols;

providing said uniform file-locking semantics in response to messages using at least one of said diverse file-locking protocols; and

enforcing said uniform file-locking semantics for all said client devices.

11. A method as in claim **10**, wherein said uniform file-locking semantics includes steps for

granting said opportunistic lock to a said first said client device in response to said first message using said first protocol; and

breaking said opportunistic lock in response to said second message using said second protocol.

12. A method as in claim **11**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

13. A method as in claim **10**, wherein said steps for enforcing said uniform file-locking semantics include steps for

granting a change-monitoring lock in response to said first message from said first client device using said first protocol; and

sending a change-notify message to said first client device in response to said second message using said second protocol.

14. A method as in claim **10**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message that attempts to violate said uniform file-locking semantics; and

responding to said selected message with an error response suited to a protocol associated with said selected message.

15. A method as in claim **10**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for obtaining a byte-range lock on a file in a selected one of said diverse file locking protocols, said byte-range lock having a lock type; and

testing whether obtaining said byte-range lock would conflict with existing locks created by messages using the same or other diverse file locking protocols.

16. A method as in claim **10**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for opening a file in a selected one of said diverse file-locking protocols, said selected message including a requested access-mode; and

testing whether opening said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file-locking protocols.

17. A method as in claim **16**,

wherein said selected message includes a requested deny-mode; and

including steps for testing whether opening said file using said requested deny-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

18. A method as in claim **10**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for reading from or writing to a file in a selected one of said diverse file locking protocols; and

testing whether reading from or writing to would conflict with existing locks created by messages using the same or other diverse file locking protocols.

19. A method as in claim **1**, wherein said steps for enforcing include steps for

receiving said first message using said first protocol, said first message being operative to lock at least a portion of a selected file;

receiving said second message using said second protocol, said second message being operative to request access to said portion;

comparing said access requested by said second message with said lock, and denying said access if prohibited by said lock.

20. A method as in claim **19**, wherein said first protocol includes CIFS.

21. A method as in claim **19**, wherein said first protocol or said second protocol includes NLM.

22. A method as in claim **19**, wherein said second protocol includes NFS.

23. A method as in claim **19**, wherein

said steps for receiving said second message include steps for recognizing said second message as being for obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and

said steps for comparing include steps for testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file-locking protocols.

24. A method as in claim **23**, wherein said steps for testing are responsive to one of said diverse file locking protocols used for said second message.

25. A method as in claim **23**, wherein said steps for testing operate at file-open time for said first protocol and at an access time for said second protocol.

26. A method as in claim **23**, wherein said steps for testing operate at file-open time for said first protocol and at a lock-request time for said second protocol.

27. A method as in claim **19**, wherein

said steps for receiving said second message include steps for recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

said steps for comparing include steps for testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

28. A method as in claim **19**, wherein

said steps for receiving said second message include steps for recognizing said second message for reading from or writing to a file using said second protocol; and

said steps for comparing include steps for testing whether accessing said file as attempted by said second message would conflict with existing locks created by messages using the same or other diverse file locking protocols.

29. A method as in claim **19**, wherein

said steps for receiving said first message include steps for granting said opportunistic lock in response to said first message; and

said steps for comparing include steps for breaking said opportunistic lock in response to said second message.

30. A method as in claim **29**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

31. A method as in claim **30**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

32. A method as in claim **1**, wherein said file-locking semantics includes a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

33. A method as in claim **1**, wherein said file-locking semantics includes

a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and

a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;

wherein said file server is responsive to comparison of said first lock mode with said second lock mode.

34. A method as in claim **33**, wherein said comparison includes a lock compatibility matrix.

35. A method as in claim **33**, wherein said comparison includes a lock conversion matrix.

36. A method as in claim **33**, wherein said second lock mode is responsive to a request for a byte-range lock.

37. A method as in claim **33**, wherein said second lock mode is responsive to a request for a NLM file lock.

38. A file server comprising a processor, mass storage, and an interface to a network, the file server operating under control of the processor to perform steps including steps for enforcing a uniform file-locking semantics among a set of client devices using a plurality of diverse file locking protocols, said file server implementing said plurality of diverse file locking protocols and enforcing said uniform file-locking semantics for said plurality of diverse file locking protocols;

wherein said uniform file-locking semantics includes steps for

granting an opportunistic lock on a selected file to a first client device in response to a first message using a first protocol; and

breaking said opportunistic lock in response to a second message using a second protocol;

wherein the first protocol and the second protocol are ones of said plurality of diverse file locking protocols.

39. A file server as in claim **38**, wherein said uniform file-locking semantics includes said opportunistic lock capable of being requested by said first client device using said first protocol; and

breaking of said opportunistic lock being triggered by a second client device using said second protocol different from said first protocol.

40. A file server as in claim **39**, wherein said first protocol includes CIFS.

41. A file server as in claim **39**, wherein said second protocol includes NFS or NLM.

42. A file server as in claim **38**, wherein said uniform file-locking semantics includes steps for

granting an opportunistic lock on a selected file to a first said client device in response to a first message using a first said protocol; and

breaking said opportunistic lock in response to a second message using a second said protocol.

43. A file server as in claim **38**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

44. A file server as in claim **38**, wherein said uniform file-locking semantics includes a change-monitoring lock type capable of being requested by said first client device using said first protocol; and

a change notification being triggered by a second client device using said second protocol different from said first protocol.

45. A file server as in claim **44**, wherein said first protocol includes CIFS.

46. A file server as in claim **44**, wherein said second protocol includes NFS.

47. A file server as in claim **38**, wherein said uniform file-locking semantics includes steps for

granting a change-monitoring lock on a selected directory to said first client device in response to said first message using said first protocol; and

sending a change-notify message to said first client device in response to said second message regarding said selected directory using said second protocol.

48. A file server as in claim **38**, wherein said steps for enforcing include steps for

recognizing said plurality of diverse file locking protocols;

providing said uniform file-locking semantics in response to messages using at least one of said diverse file locking protocols; and

enforcing said uniform file-locking semantics for all said client devices.

49. A file server as in claim **48**, wherein said uniform file-locking semantics includes steps for

granting said opportunistic lock to said first said client device in response to said first message using said first protocol; and

breaking said opportunistic lock in response to said second message using said second protocol.

50. A file server as in claim **49**, wherein said steps for breaking include steps for sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

51. A file server as in claim **48**, wherein said steps for enforcing said uniform file-locking semantics include steps for

granting a change-monitoring lock in response to said first message from said first client device using said first protocol; and

sending a change-notify message to said first client device in response to said second message using said second protocol.

52. A file server as in claim **48**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message that attempts to violate said uniform file-locking semantics; and

responding to said selected message with an error response suited to a protocol associated with said selected message.

53. A file server as in claim **48**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for obtaining a byte-range lock on a file in a selected one of said diverse file locking protocols, said byte-range lock having a lock type; and

testing whether obtaining said byte-range lock would conflict with existing locks created by messages using the same or other diverse file locking protocols.

54. A file server as in claim **48**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for opening a file in a selected one of said diverse file locking protocols, said selected message including a requested access-mode; and

testing whether opening said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

55. A file server as in claim **54**,

wherein said selected message includes a requested deny-mode; and

including steps for testing whether opening said file using said requested deny-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

56. A file server as in claim **48**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for reading from or writing to a file in a selected one of said diverse file locking protocols; and

testing whether reading from or writing to would conflict with existing locks created by messages using the same or other diverse file locking protocols.

57. A file server as in claim **38**, wherein said steps for enforcing include steps for

receiving said first message using said first protocol, said first message being operative to lock at least a portion of a selected file;

receiving said second message using said second protocol, said second message being operative to request access to said portion;

comparing said access requested by said second message with said lock, and denying said access if prohibited by said lock.

58. A file server as in claim **57**, wherein said first protocol includes CIFS.

59. A file server as in claim **57**, wherein said first protocol or said second protocol includes NLM.

60. A file server as in claim **57**, wherein said second protocol includes NFS.

61. A file server as in claim **57**, wherein

said steps for receiving said second message include steps for recognizing said second message as being for obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and

said steps for comparing include steps for testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file locking protocols.

62. A file server as in claim **61**, wherein said steps for testing are responsive to one of said diverse file locking protocols used for said second message .

63. A file server as in claim **61**, wherein said steps for testing operate at file-open time for said first protocol and at an access time for said second protocol.

64. A file server as in claim **61**, wherein said steps for testing operate at file-open time for said first protocol and at a lock-request time for said second protocol.

65. A file server as in claim **57**, wherein

said steps for receiving said second message include steps for recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

said steps for camparing include steps for testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

66. A file server as in claim **57**, wherein

said steps for receiving said second message include steps for recognizing said second message for reading from or writing to a file using said second protocol; and

said steps for comparing include steps for testing whether accessing said file as attempted by said second message would conflict with existing locks vreated by messages using the same or other diverse file locking protocols.

67. A file server as in claim **57**, wherein

said steps for receiving said first message include steps for granting said opportunistic lock in response to said first message; and

said steps for comparing include steps for breaking said opportunistic lock in response to said second message.

68. A file server as in claim **67**, wherein said steps for breaking include steps for sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

69. A file server as in claim **68**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

70. A file server as in claim **38**, wherein said file-locking semantics includes a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

71. A file server as in claim **38**, wherein said file-locking semantics includes

a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and

a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;

wherein said file server is responsive to comparison of said first lock mode with said second lock mode.

72. A file server as in claim **71**, wherein said comparison includes a lock compatibility matrix.

73. A file server as in claim **71**, wherein said comparison includes a lock conversion matrix.

74. A file server as in claim **71**, wherein said second lock mode is responsive to a request for a byte-range lock.

75. A file server as in claim **71**, wherein said second lock mode is responsive to a request for a NLM file lock.

76. A memory storing information including instructions, the instructions executable by a processor to operate a file server, the instructions comprising steps for enforcing file-locking semantics among a set of client devices using a plurality of diverse file locking protocols, said file server implementing said plurality of diverse file locking protocols and enforcing said uniform file-locking semantics for said plurality of diverse file locking protocols;

wherein said uniform file-locking semantics includes steps for

granting an opportunistic lock on a selected file to a first client device in response to a first message using a first protocol; and

breaking said opportunistic lock in response to a second message using a second protocol;

wherein the first protocol and the second protocol are ones of said plurality of diverse file locking protocols.

77. A memory as in claim **76**, wherein said uniform file-locking semantics includes said opportunistic lock capable of being requested by said first client device using said first protocol; and

breaking of said opportunistic lock being triggered by a second client device using said second protocol different from said first protocol.

78. A memory as in claim **77**, wherein said first protocol includes CIFS.

79. A memory as in claim **77**, wherein said second protocol includes NFS or NLM.

80. A memory as in claim **76**, wherein said uniform file-locking semantics includes steps for

granting an opportunistic lock on a selected file to a first said client device in response to a first message using a first said protocol; and

breaking said opportunistic lock in response to a second message using a second said protocol.

81. A memory as in claim **76**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

82. A memory as in claim **76**, wherein said uniform file-locking semantics includes a change-monitoring lock type capable of being requested by a first client device using a first protocol; and

a change notification being triggered by a second client device using a second protocol different from said first protocol.

83. A memory as in claim **82**, wherein said first protocol includes CIFS.

84. A memory as in claim **82**, wherein said second protocol includes NFS.

85. A memory as in claim **76**, wherein said uniform file-locking semantics includes steps for

granting a change-monitoring lock on a selected directory to said first client device in response to said first message using said first protocol; and

sending a change-notify message to said first client device in response to said second message regarding said selected directory using said second protocol.

86. A memory as in claim **76**, wherein said steps for enforcing include steps for

recognizing said plurality of diverse file locking protocols;

providing said uniform file-locking semantics in response to messages using at least one of said diverse file locking protocols; and

enforcing said uniform file-locking semantics for all said client devices.

87. A memory as in claim **86**, wherein said uniform file-locking semantics includes steps for

granting said opportunistic lock to said first said client device in response to said first message using said first protocol; and

breaking said opportunistic lock in response to said second message using said second protocol.

88. A memory as in claim **87**, wherein said steps for breaking include steps for

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

89. A memory as in claim **86**, wherein said steps for enforcing said uniform file-locking semantics include steps for

granting a change-monitoring lock in response to said first message from said first client device using said first protocol; and

sending a change-notify message to said first client device in response to said second message using said second protocol.

90. A memory as in claim **86**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message that attempts to violate said uniform file-locking semantics; and

responding to said selected message with an error response suited to a protocol associated with said selected message.

91. A memory as in claim **86**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for obtaining a byte-range lock on a file in a selected one of said diverse file locking protocols, said byte-range lock having a lock type; and

testing whether obtaining said byte-range lock would conflict with existing locks created by messages using the same or other diverse file locking protocols.

92. A memory as in claim **86**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for opening a file in a selected one of said diverse file locking protocols, said selected message including a requested access-mode; and

testing whether opening said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

93. A memory as in claim **92**,

wherein said selected message includes a requested deny-mode; and

including steps for testing whether opening said file using said requested deny-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

94. A memory as in claim **86**, wherein said steps for enforcing said uniform file-locking semantics include steps for

recognizing a selected message for reading from or writing to a file in a selected one of said diverse file locking protocols; and

testing whether reading from or writing to would conflict with existing locks created by messages using the same or other diverse file locking protocols.

95. A memory as in claim **76**, wherein said steps for enforcing include steps for

receiving said first message using said first protocol, said first message being operative to lock at least a portion of a selected file;

receiving said second message using said second protocol, said second message being operative to request access to said portion;

comparing said access requested by said second message with said lock, and denying said access if prohibited by said lock.

96. A memory as in claim **95**, wherein said first protocol includes CIFS.

97. A memory as in claim **95**, wherein said first protocol or said second protocol includes NLM.

98. A memory as in claim **95**, wherein said second protocol includes NFS.

99. A memory as in claim **95**, wherein

said steps for receiving said second message include steps for recognizing said second message as being for

obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and

said steps for comparing include steps for testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file locking protocols.

100. A memory as in claim **99**, wherein said steps for testing are responsive to one of said diverse file locking protocols used for said second message.

101. A memory as in claim **99**, wherein said steps for testing operate at file-open time for said first protocol and at an access time for said second protocol.

102. A memory as in claim **99**, a method as in claim **24**, wherein said steps for testing operate at file-open time for said first protocol and at a lock-request time for said second protocol.

103. A memory as in claim **95**, wherein

said steps for receiving said second message include steps for recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

said steps for comparing include steps for testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

104. A memory as in claim **95**, wherein

said steps for receiving said second message include steps for recognizing said second message for reading from or writing to a file using said second protocol; and

said steps for comparing include steps for testing whether accessing said file as attempted by said second message would conflict with existing locks created by messages using the same or other diverse file locking protocols.

105. A memory as in claim **95**, wherein

said steps for receiving said first message include steps for granting said opportunistic lock in response to said first message; and

said steps for comparing include steps for breaking said opportunistic lock in response to said second message.

106. A memory as in claim **105**, wherein said steps for breaking include steps for sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

107. A memory as in claim **106**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

108. A memory as in claim **76**, wherein said file-locking semantics includes a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

109. A memory as in claim **76**, wherein said file-locking semantics includes

a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and

a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;

wherein said file server is responsive to comparison of said first lock mode with said second lock mode.

110. A memory as in claim **109**, wherein said comparison includes a lock compatibility matrix.

111. A memory as in claim **109**, wherein said comparison includes a lock conversion matrix.

112. A memory as in claim **109**, wherein said second lock mode is responsive to a request for a byte-range lock.

113. A memory as in claim **109**, wherein said second lock mode is responsive to a request for a NLM file lock.

114. A file server comprising processing means, mass storage means for storing files, and interface means for interfacing to a network, the processing means for enforcing a uniform file-locking semantics among a set of client devices using a plurality of diverse file locking protocols and enforcing said uniform file-locking semantics for said plurality of diverse file locking protocols;

wherein said uniform file-locking semantics includes steps for

granting am opportunistic lock on a selected file to a first client device in response to a first message using a first protocol; and

breaking said opportunistic lock in response to a second message using a second protocol;

wherein the first protocol and the second protocol are ones of said plurality of diverse file locking protocols.

* * * * *

Exhibit B

US007293097B2

(12) **United States Patent**
Borr

(10) **Patent No.: US 7,293,097 B2**
(45) **Date of Patent: *Nov. 6, 2007**

(54) **ENFORCING UNIFORM FILE-LOCKING FOR DIVERSE FILE-LOCKING PROTOCOLS**

(75) Inventor: **Andrea Borr**, La Jolla, CA (US)

(73) Assignee: **Network Appliance, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 874 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/230,877**

(22) Filed: **Aug. 28, 2002**

(65) **Prior Publication Data**

US 2003/0065796 A1 Apr. 3, 2003

Related U.S. Application Data

(63) Continuation of application No. 09/176,599, filed on Oct. 21, 1998, now Pat. No. 6,516,351, which is a continuation of application No. 08/985,398, filed on Dec. 5, 1997, now abandoned, application No. 10/230,877, which is a continuation of application No. PCT/US98/25388, filed on Nov. 30, 1998.

(51) **Int. Cl.**
G06F 15/16 (2006.01)
G06F 7/00 (2006.01)

(52) **U.S. Cl.** **709/229**; 707/1

(58) **Field of Classification Search** 710/200; 709/229; 707/1
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,719,569	A	1/1988	Ludemann et al.
4,742,450	A	5/1988	Duvall et al.
4,780,821	A	10/1988	Crossley
4,825,354	A	4/1989	Agrawal et al.
4,887,204	A	12/1989	Johnson et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0 306 244	3/1989

(Continued)

OTHER PUBLICATIONS

"Mapping the VM Text Files to the AIX Text Files," IBM Technical Disclosure Bulletin, vol. 33, No. 2, Jul. 1990, p. 341 XP000123641, IBM Corp., New York, US ISSN: 0018-8689—the whole document.

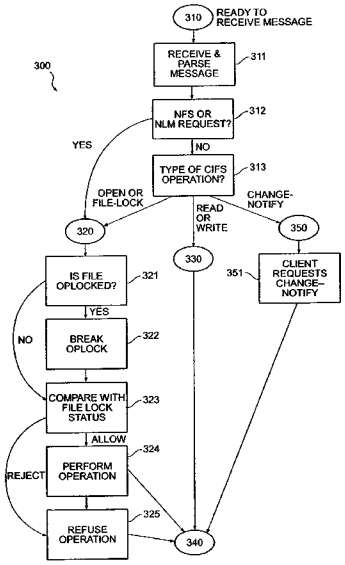
(Continued)

Primary Examiner—David Wiley
Assistant Examiner—J. Bret Dennison
(74) *Attorney, Agent, or Firm*—Blakely Sokoloff Taylor & Zafman LLP

(57) **ABSTRACT**

The invention provides a method and system for connect interoperation of multiple diverse file server or file locking protocols, using a uniform multi-protocol lock management system. A file server determines, before allowing any client device to access data or to obtain a lock, whether that would be inconsistent with existing locks, regardless of originating client device or originating protocol for those existing locks. A first protocol enforces mandatory file-open and file-locking together with an opportunistic file-locking technique, while a second protocol lacks file-open semantics and provides only for advisory byte-range and file locking.

79 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

4,937,763	A		6/1990	Mott
4,984,272	A		1/1991	McIlroy et al.
5,008,786	A		4/1991	Thatte
5,043,876	A		8/1991	Terry
5,067,099	A		11/1991	McCown et al.
5,113,442	A		5/1992	Moir
5,144,659	A		9/1992	Jones
5,146,588	A		9/1992	Crater et al.
5,202,983	A		4/1993	Orita et al.
5,222,217	A		6/1993	Blount et al.
5,261,051	A		11/1993	Masden et al.
5,283,830	A		2/1994	Hinsley et al.
5,319,780	A		6/1994	Catino et al.
5,335,235	A		8/1994	Arnott
5,504,883	A		4/1996	Coverston et al.
5,535,375	A	*	7/1996	Eshel et al. 703/27
5,572,711	A		11/1996	Hirsch et al.
5,596,754	A		1/1997	Lomet
5,604,862	A		2/1997	Midgely et al.
5,617,568	A		4/1997	Ault et al.
5,628,005	A	*	5/1997	Hurvig 707/8
5,649,152	A		7/1997	Ohran et al.
5,649,196	A		7/1997	Woodhill et al.
5,668,958	A		9/1997	Bendert et al.
5,675,726	A		10/1997	Hohenstein et al.
5,675,782	A		10/1997	Montague et al.
5,689,701	A		11/1997	Ault et al.
5,720,029	A		2/1998	Kern et al.
5,721,916	A		2/1998	Pardikar
5,737,523	A		4/1998	Callaghan et al.
5,737,744	A		4/1998	Callison et al.
5,740,367	A		4/1998	Spilo
5,742,752	A		4/1998	Dekoning
5,761,669	A		6/1998	Montague et al.
5,819,292	A		10/1998	Hitz et al.
5,819,310	A		10/1998	Vishlitzky
5,825,877	A		10/1998	Dan et al.
5,828,876	A	*	10/1998	Fish et al. 707/1
5,835,953	A		11/1998	Ohran
5,876,278	A		3/1999	Cheng
5,890,959	A		4/1999	Pettit et al.
5,915,087	A		6/1999	Hammond et al.
5,931,935	A		8/1999	Calbrera et al.
5,940,828	A		8/1999	Anaya et al.
5,948,110	A		9/1999	Hitz et al.
5,963,962	A		10/1999	Hitz et al.
5,996,086	A		11/1999	Delaney et al.
5,999,943	A		12/1999	Nori et al.
6,026,474	A	*	2/2000	Carter et al. 709/203
6,067,541	A		5/2000	Raju et al.
6,085,234	A		7/2000	Pitts et al.
6,275,953	B1		8/2001	Vahalia et al.
6,279,011	B1		8/2001	Muhlestein
6,457,130	B2		9/2002	Hitz et al.
6,516,351	B2	*	2/2003	Borr 709/229
2002/0019874	A1		2/2002	Borr
2002/0019936	A1	*	2/2002	Hitz et al. 713/165

FOREIGN PATENT DOCUMENTS

EP	0 308 056	3/1989
EP	359384	3/1990
EP	0 410 630	1/1991
EP	453193	10/1991
EP	0 477 039	3/1992
EP	0 537 098	4/1993
EP	0 566 967	10/1993
EP	629 956	12/1994
EP	747 829	12/1996
EP	756 235	1/1997
EP	0 760 503	3/1997
JP	1-273395	11/1989
NI	134938	10/2001
WO	WO 89/03 086	4/1989
WO	WO 98/21656	5/1998
WO	WO 99/30254	6/1999
WO	WO 99/30254 A1	6/1999
WO	WO 99/45456	9/1999
WO	WO 99/45456 A1	9/1999
WO	WO 99/66 401	12/1999
WO	WO 01/31 446	5/2001
WO	WO 02/29572	4/2002

OTHER PUBLICATIONS

"Migrated Data Backup Utility," IBM Technical Disclosure Bulletin, vol. 37, No. 06B, Jun. 1994, pp. 505-507, XP000456079, IBM Corp. New York, US ISSN: 0018-8689.

Tanner, J., "CIFS: Common Internet File System," Unix Review, vol. 31, Feb. 1997, pp. 31/32, 34, XPOOO783952.

Reichel, R., "Inside Windows NT Security: Part 1," Windows/DOS Developers' Journal, vol. 4., No. 4, Apr. 1993, pp. 6-19, XP002107445, Lawrence, KS, USA.

Srinivasan, B., et al., "Recoverable File System for Microprocessor systems" Microprocessors and Microsystems, vol. 9, No. 4,May 1985, London, GB, pp. 179-183, XP002031805.

Nass, Richard, "SCSI Disk-Array Controller Board Handles RAID and Can Pass Data t 40 Mbytes/S.—Connect Disk Arrays to EISA or PCI Buses," Electronic Design, vol. 41, No. 28, Nov. 1993, Cleveland, OH, USA, pp. 152-154, XP000417908.

Andrea J. Borr. "Secure Share: Safe UNIX/Windows File Sharing through Multiprotocol Locking". 2nd. USENIX Windows NT Symposium, USENIX Assocation pp. 117-126, XP002097387, Aug. 28, 2002.

John Row. "Lan Software Links Diverse Machines. OS's". Mini-Micro Systems, Sep. 1985.

* cited by examiner

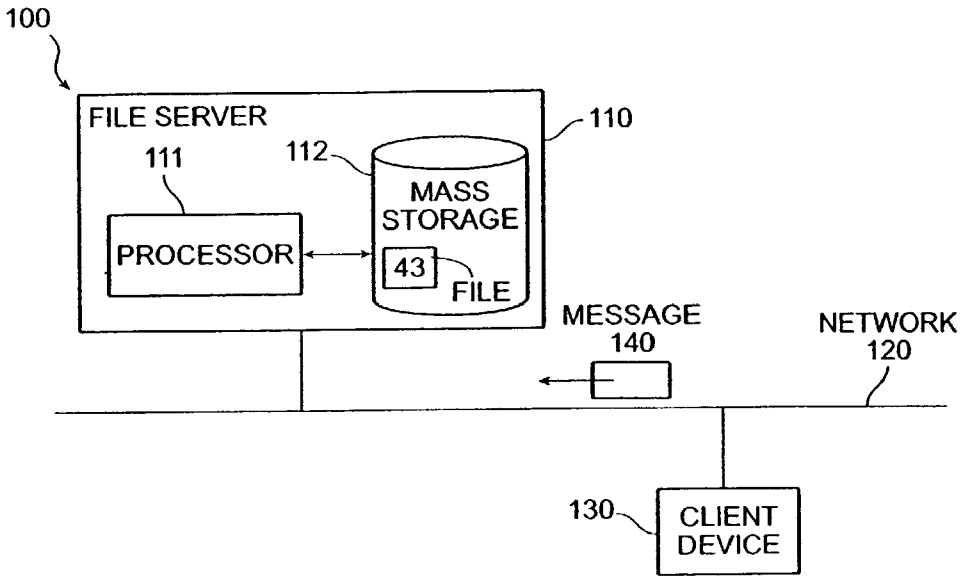

100
FILE SERVER
110
111
112
MASS STORAGE
PROCESSOR
43
FILE
MESSAGE
140
NETWORK
120
130
CLIENT DEVICE

FIG. 1

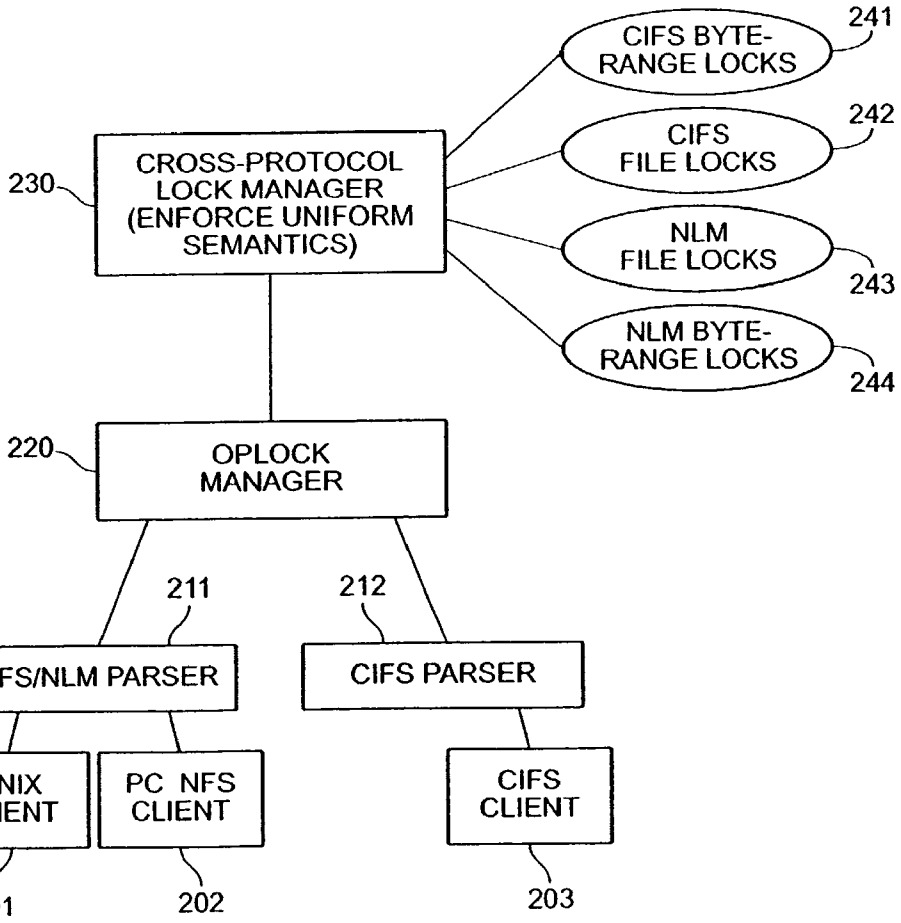

241
CIFS BYTE-RANGE LOCKS
242
CIFS FILE LOCKS
230
CROSS-PROTOCOL LOCK MANAGER (ENFORCE UNIFORM SEMANTICS)
NLM FILE LOCKS
243
NLM BYTE-RANGE LOCKS
244
220
OPLOCK MANAGER
211
212
NFS/NLM PARSER
CIFS PARSER
UNIX CLIENT
PC NFS CLIENT
CIFS CLIENT
201
202
203

FIG. 2

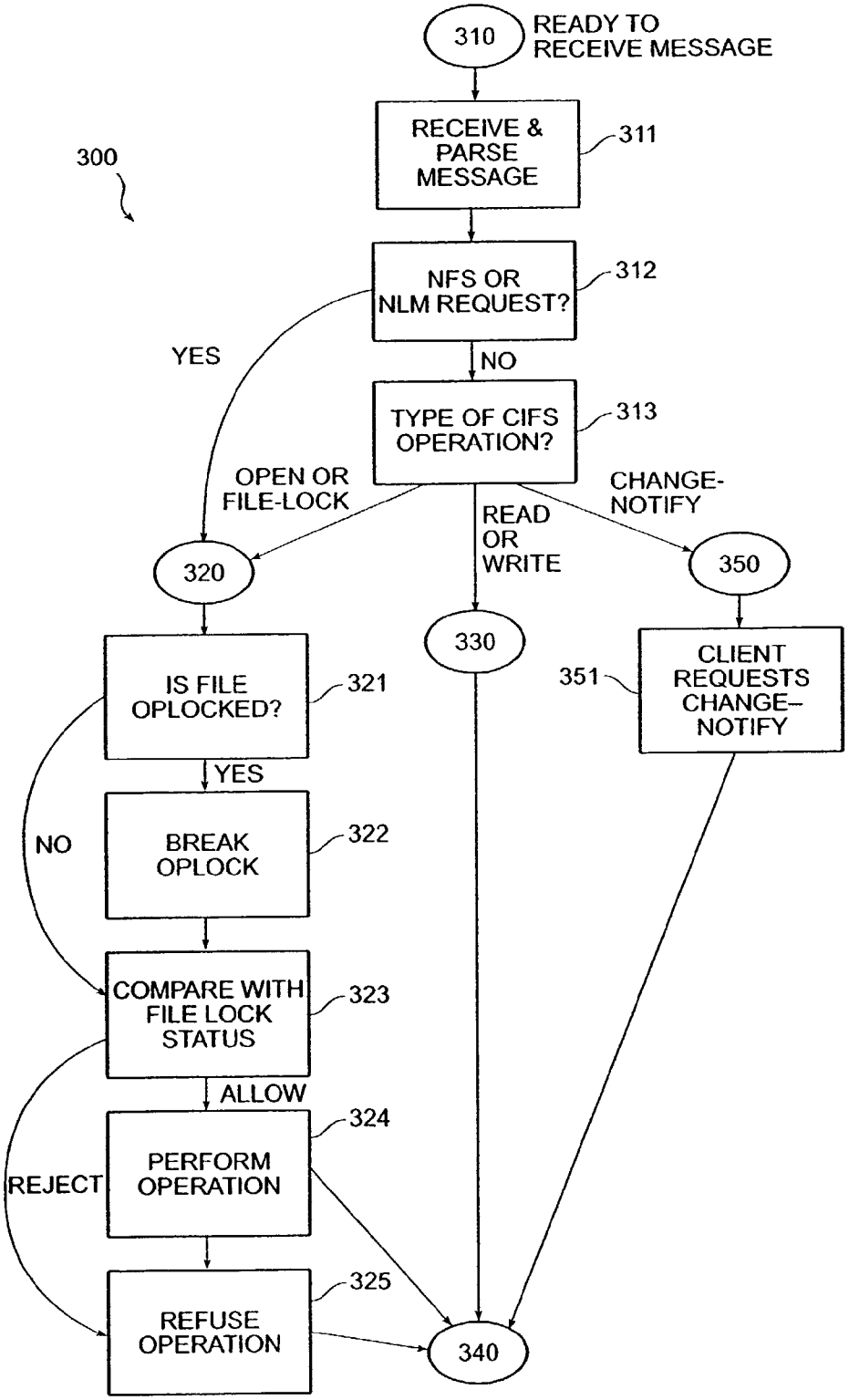

300
310
READY TO RECEIVE MESSAGE
RECEIVE & PARSE MESSAGE
311
NFS OR NLM REQUEST?
312
YES
NO
TYPE OF CIFS OPERATION?
313
OPEN OR FILE-LOCK
READ OR WRITE
CHANGE-NOTIFY
320
330
350
IS FILE OPLOCKED?
321
YES
NO
BREAK OPLOCK
322
COMPARE WITH FILE LOCK STATUS
323
ALLOW
REJECT
PERFORM OPERATION
324
REFUSE OPERATION
325
340
351
CLIENT REQUESTS CHANGE–NOTIFY

FIG. 3

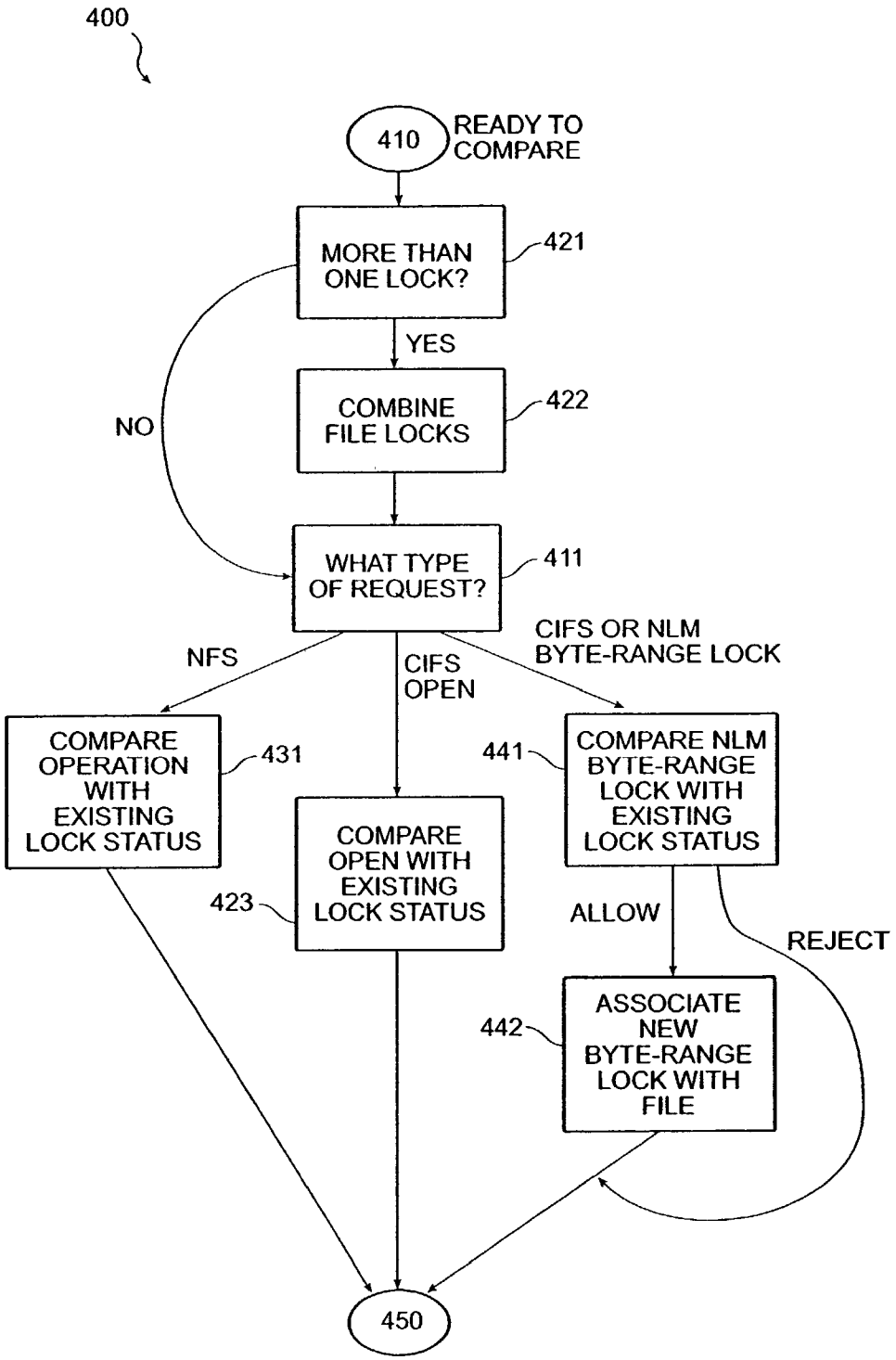

400
410
READY TO COMPARE
MORE THAN ONE LOCK?
421
YES
NO
COMBINE FILE LOCKS
422
WHAT TYPE OF REQUEST?
411
NFS
CIFS OPEN
CIFS OR NLM BYTE-RANGE LOCK
COMPARE OPERATION WITH EXISTING LOCK STATUS
431
COMPARE OPEN WITH EXISTING LOCK STATUS
423
COMPARE NLM BYTE-RANGE LOCK WITH EXISTING LOCK STATUS
441
ALLOW
REJECT
ASSOCIATE NEW BYTE-RANGE LOCK WITH FILE
442
450

FIG. 4

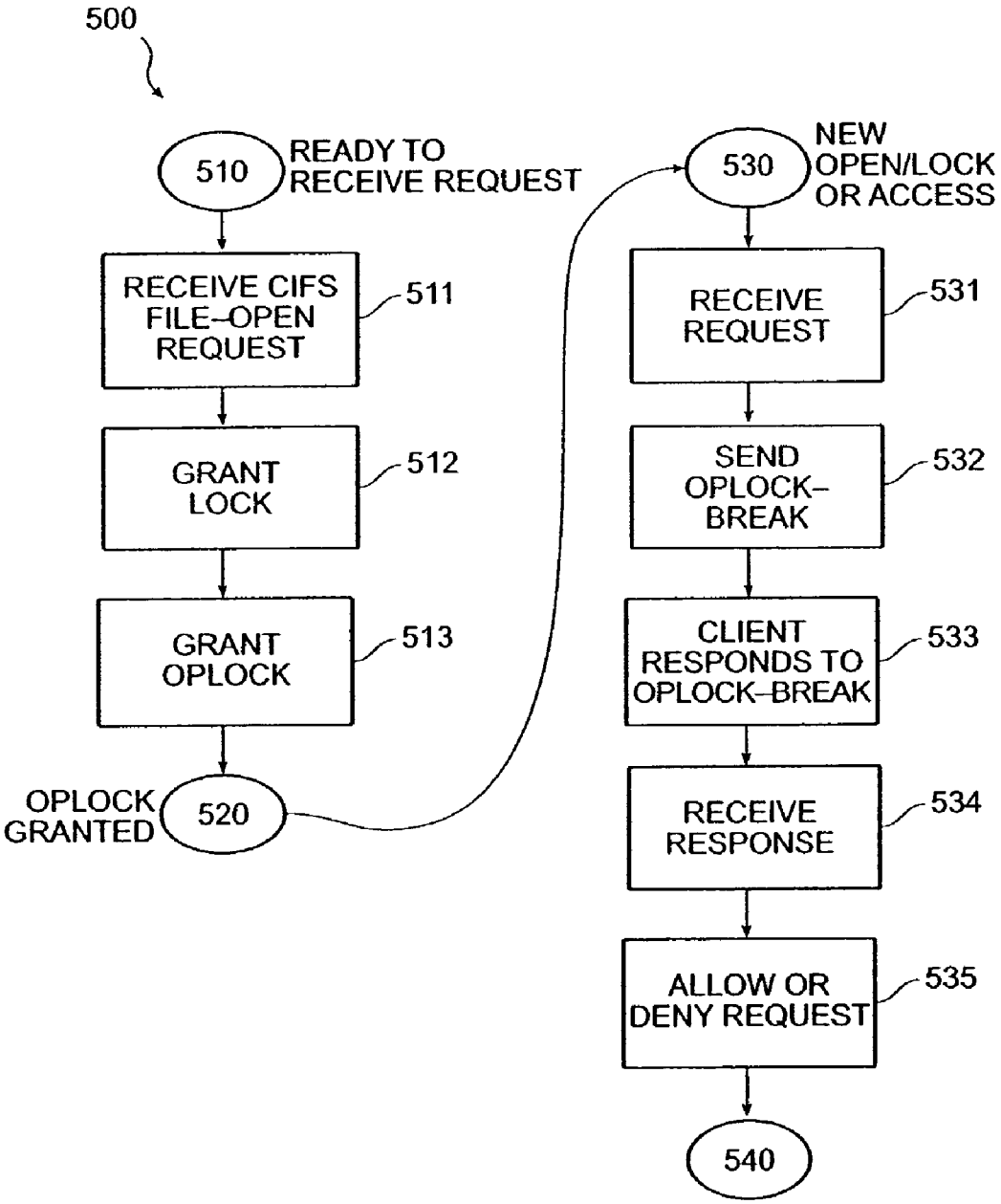

500
510
READY TO RECEIVE REQUEST
RECEIVE CIFS FILE-OPEN REQUEST
511
GRANT LOCK
512
GRANT OPLOCK
513
OPLOCK GRANTED
520
530
NEW OPEN/LOCK OR ACCESS
RECEIVE REQUEST
531
SEND OPLOCK-BREAK
532
CLIENT RESPONDS TO OPLOCK-BREAK
533
RECEIVE RESPONSE
534
ALLOW OR DENY REQUEST
535
540

FIG. 5

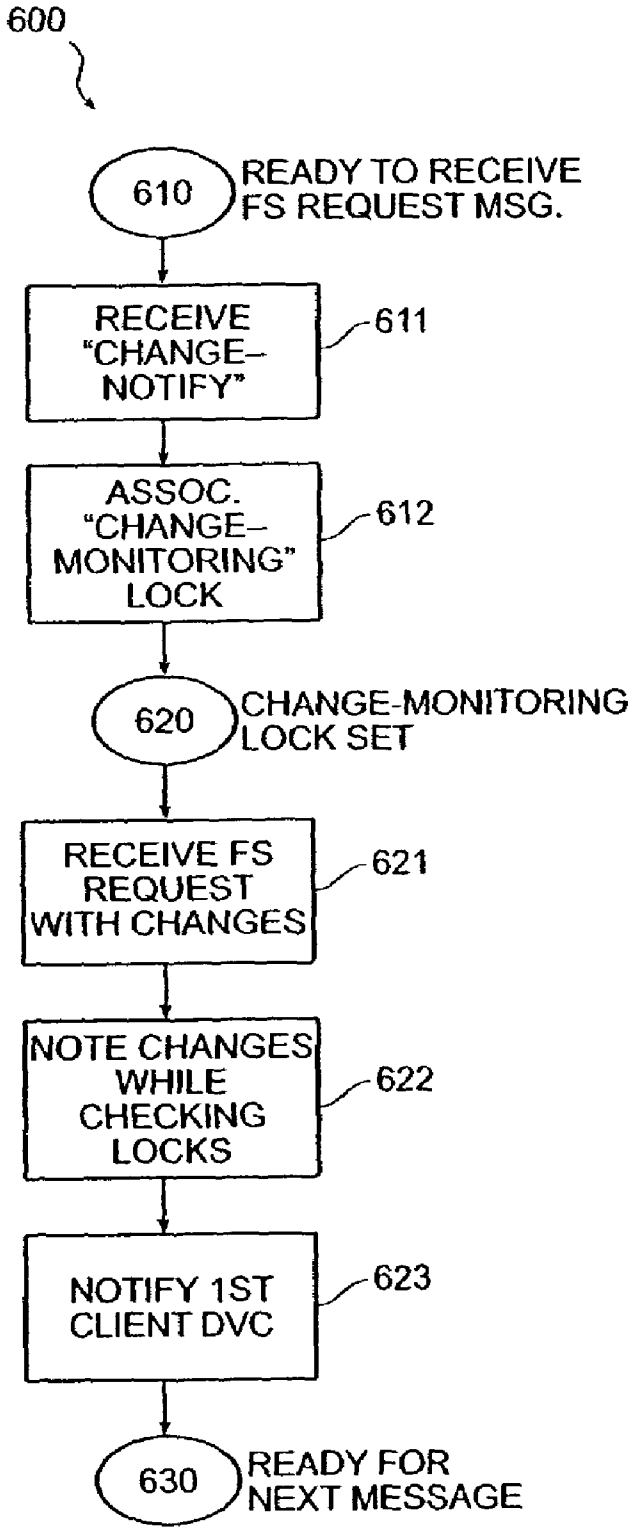

600
610
READY TO RECEIVE FS REQUEST MSG.
RECEIVE "CHANGE–NOTIFY"
611
ASSOC. "CHANGE–MONITORING" LOCK
612
620
CHANGE-MONITORING LOCK SET
RECEIVE FS REQUEST WITH CHANGES
621
NOTE CHANGES WHILE CHECKING LOCKS
622
NOTIFY 1ST CLIENT DVC
623
630
READY FOR NEXT MESSAGE

FIG. 6

ENFORCING UNIFORM FILE-LOCKING FOR DIVERSE FILE-LOCKING PROTOCOLS

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a continuation of application Ser. No. 09/176,599, filed Oct. 21, 1998 now U.S. Pat. No. 6,516,351, which is a continuation of application Ser. No. 08/985,398, filed Dec. 5, 1997 (now abandoned). This application also is a continuation of PCT application Ser. No. PCT/US98/25388 filed Nov. 30, 1998.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to locking in a multi-protocol file server.

2. Related Art

In an integrated computer network, it is desirable for multiple client devices to share files. One known method is to provide a network file server for storing files, capable of receiving and responding to file server requests from those client devices. These file server requests are made using a file server protocol, which is recognized and adhered to by both the file server and the client device. Because the files are stored at the file server, multiple client devices have the opportunity to share simultaneous access to files.

One problem in the art is that there are multiple diverse file server protocols, each with differing semantics for file operations. It is known to provide a single file server that recognizes multiple file server protocols, but there is a particular difficulty in that many file server protocols have differing and incompatible semantics for file locking and file sharing. Incompatible locking semantics presents a hurdle in providing a single file system to multiple diverse client devices. If a first client device relies on a first file server protocol (having first file-locking semantics), a second client device using a second file server protocol (having different file-locking semantics) can cause applications at that first client device to fail catastrophically. Thus, correct operation of each file server protocol relies on conformity with its file-locking semantics by all other file server protocols.

For example, one protocol commonly used for devices using the Unix operating system (or a variant thereof) is the NFS ("Network File System") protocol. Devices using the Windows operating system (or a variant thereof) can also use the NFS protocol by means of the "PC NFS" implementation. The NFS protocol is designed to be stateless, and so does not provide any semantics for files to be locked against sharing or otherwise restricted to a single client device. In contrast, one protocol commonly used for devices using the Windows operating system is the CIFS ("Common Internet File System") protocol. The CIFS protocol has an extensive mandatory file-locking semantics, which CIFS client devices rely on and expect to be adhered to.

In known systems, the NFS protocol has been augmented with an adjunct file-locking protocol, NLM ("Network Lock Manager"), but the NFS protocol treats NLM locks as merely advisory. While this method achieves the purpose of providing file-locking semantics to those NFS applications that use it, it does not prevent NFS applications from ignoring those file-locking semantics, nor does it allow client devices to rely on the file-locking semantics of multiple diverse file server protocols.

Accordingly, it would be desirable to provide a method and system for enforcing file-locking semantics among

client devices using multiple diverse file server protocols. This advantage is achieved in an embodiment of the invention in which a uniform set of file-locking semantics is integrated into the kernel of a multi-protocol file server and enforced for client devices using any of the diverse file server protocols recognized by the server. In a preferred embodiment, specific file-locking semantics of the CIFS protocol are implemented so as to allow NFS client devices to inter-operate with CIFS client devices so as to protect data integrity during client access to a shared file system resident on a network file server.

SUMMARY OF INVENTION

The invention provides a method and system for correct interoperation of multiple diverse file server protocols. A file server recognizing multiple diverse file server protocols provides a uniform multi-protocol lock management system (including a uniform file-locking semantics), which it enforces for all client devices and all recognized file server protocols. In a preferred embodiment, a first file server protocol (such as CIFS) enforces-mandatory file-open and file-locking semantics together with an opportunistic file-locking technique, while a second file server protocol (such as NFS, together with an adjunct protocol NLM) lacks file-open semantics and provides only for advisory byte-range and file locking semantics.

The uniform file-locking semantics provides that the file server determines, before allowing any client device to read or write data, or to obtain a new file lock or byte-range lock, whether that would be inconsistent with existing locks, regardless of originating client device and regardless of originating file server protocol or file-locking protocol for those existing locks. In the case of CIFS client devices attempting to read or write data, the file server performs this check when the client device opens the file. In the case of CIFS client devices requesting a byte-range lock, the file server performs this check when the client device requests the byte-range lock. In the case of NFS client devices, the file server performs this check when the client device actually issues the read or write request, or when the NFS client device requests an NLM byte-range lock indicating intent to read or write that byte range. Enforcing file-locking semantics protects file data against corruption by NFS client devices.

In a second aspect of the invention, a CIFS client device, upon opening a file, can obtain an "oplock" (opportunistic lock), an exclusive file lock that permits only that one client to read or write the file. When a client device issues a non-CIFS (that is, NFS or NLM) protocol request for the oplocked file, the file server sends an oplock-break message to the CIFS client device, giving the CIFS client device the opportunity to flush any cached write operations and possibly close the file. Allowing NFS and NLM requests to break oplocks ensures that file data remains available to NFS client devices simultaneously with protecting integrity of that file data.

In a third aspect of the invention, a CIFS client device can obtain a "change-monitoring" lock for a directory in the file system, so as to be notified by the file server whenever there is a change to that directory. (Changes to a directory include creating, deleting or renaming files within that directory, or moving files into or out of that directory.) The file server notes changes to the directory by both CIFS client devices and non-CIFS client devices, and notifies those CIFS client devices with "change-monitoring" locks of those changes.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a first block diagram of a system including a multi-protocol file server.

FIG. **2** shows a second block diagram of a system including a multi-protocol file server.

FIG. **3** shows a process flow diagram of a method of operating a multi-protocol file server.

FIG. **4** shows a process flow diagram of a method of operating a cross-protocol lock manager in a multi-protocol file server.

FIG. **5** shows a process flow diagram of a method of operating an oplock manager in a multi-protocol file server.

FIG. **6** shows a process flow diagram of a method of operating a change-notify manager in a multi-protocol file server.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. Those skilled in the art would recognize after perusal of this application that embodiments of the invention can be implemented using general purpose processors or special purpose processors or other circuits adapted to particular process steps and data structures described herein, and that implementation of the process steps and data structures described herein would not require undue experimentation or further invention.

System Architecture (Client/Server)

FIG. **1** shows a first block diagram of a system including a multi-protocol file server.

A system **100** includes a file server **110**, a computer network **120**, and a plurality of client devices **130**.

The file server **110** includes a processor **111** and mass storage **112**. The mass storage **112** is capable of storing and retrieving a set of files **113** having data for storage and retrieval. The processor **111** is capable of receiving a sequence of request messages **140** from the network **120**, parsing those messages as commands and data, and manipulating the files **113** on the mass storage **112**, and sending response messages, in response to those commands and data.

The file server **110** and the client devices **130** are coupled to the network **120** and communicate using messages **140** transmitted on the network **120**. The messages **140** include file system requests transmitted by the client devices **130** to the file server **110** and file system responses transmitted by the file server **110** to the client devices **130**.

System Architecture (File-locking Semantics)

FIG. **2** shows a second block diagram of a system including a multi-protocol file server.

The system **100** includes the set of client devices **130**, including Unix client devices **201**, PC NFS Windows client devices **202**, and CIFS Windows client devices **203**. Unix client devices **201** execute the Unix operating system and use the Unix/NFS file server protocol. PC NFS Windows client devices **202** execute the Windows operating system and use the PC NFS file server protocol. CIFS Windows client devices **203** execute the Windows operating system and use the CIFS file server protocol.

Unix client devices **201** and PC NFS Windows client devices **202** communicate with the file server **110** using the NFS file server protocol, which is recognized at the file server **110** by an NFS file server protocol parser **211**. CIFS Windows client devices **203** communicate with the file server **110** using the CIFS file server protocol, which is recognized at the file server **110** by a CIFS file server protocol parser **212**. Messages using either the NFS file server protocol or the CIFS file server protocol are parsed by the processor **111** and processed by an oplock manager **220**.

The oplock manager **220** manages access to files **113** having CIFS oplocks. Its operation is described in further detail with regard to FIG. **3** and FIG. **5**. The oplock manager element **220** is coupled to a cross-protocol lock manager **230**.

The cross-protocol lock manager **230** manages the tile-locking semantics of the file server **110**. It processes and records information regarding four types of locks—CIFS byte-range locks **241**, CIFS file locks **242**, PC NFS (NLM) file locks **243**, and NLM byte-range locks **244**. The operation of the cross-protocol lock manager **230** in enforcing uniform file-locking semantics is described in further detail with regard to FIG. **3** and FIG. **4**.

Differing File-locking Semantics

As noted with regard to FIG. **2**, file server request messages **140** can be received from Unix client devices **201**, PC NFS Windows client devices **202**, or CIFS Windows client devices **203**, and can use the NFS file server protocol or the CIFS file server protocol. In addition to each using differing file server protocols, each of these types of client device **130** has a different model of file locking provided by the file server **110**.

In particular, the NFS file server protocol provides for performing file system operations without any form of file-open or file-close semantics. These NFS file system operations can include read or write operations to file data, or file system manipulation (that is, read and write operations to directories). File system manipulation can include creating files or directories, renaming files or directories, moving files from one directory to another, or removing (deleting) files or directories from the file system.

The NLM adjunct protocol provides for obtaining and releasing byte-range locks for files. These byte-range locks can be "read locks," which induce other compliant applications (such as at other client devices **130**) to refrain from writing to the specified byte-range. These byte-range locks can alternatively be "write locks," which induce other compliant applications to refrain from either reading from or writing to the specified byte-range.

The CIFS file server protocol provides for performing file-open operations, and obtaining file locks on the files **113** to be opened, before attempting any read or write operations to data in those files **113**. At file-open time, a CIFS client device **130** can specify the access-mode it desires (read-only, write-only, or read-write), and the deny-mode it desires to impose on other client devices **130** attempting to open the same file **113** (deny-none, deny-read, deny-write, or deny-all). Thereafter, CIFS file system operations need only be

checked against the access-mode that the file-open obtained. A CIFS client device **130** can also specify a byte-range lock for a byte-range in a file the client device **130** holds open. The byte-range lock is either an exclusive lock, also called a "write lock" (having access-mode read-write and deny-mode deny-all), or a nonexclusive lock, also called a "read lock" (having access-mode read-only and deny-mode deny-write).

The file server **110** determines a lock mode that combines the access-mode and the deny-mode. As used herein, the phrase "lock mode" refers to a uniform lock mode imposed by the file server **110** which combines an access-mode and a deny-mode.

At file-open time, CIFS client devices **130** can also obtain an oplock (opportunistic lock), which provides that the CIFS client device **130** opening the file has exclusive access to the file so long as another client device **130** does not attempt to use the file. The oplock provides a higher degree of exclusivity to the file than strictly necessary for the client device **130**, with the caveat that the exclusivity of the oplock can be broken by attempted access by another client device **130**.

The file server **110** provides for correct interoperation among client devices **130** using NFS (with or without the adjunct protocol NLM) or CIFS. To provide for correct interoperation, the file server **110** provides a uniform file-locking semantics. In a preferred embodiment, the uniform file-locking semantics has the following effects:

- The file server **110** prevents Unix client devices **201** from performing NFS write operations that would overwrite data in a file **113** that is already opened and in use by a CIFS client with deny-modes deny-write or deny-all.
- The file server **110** prevents Unix client devices **201** and PC NFS Windows client devices **202** from performing NFS file system operations that would remove or rename a file **113** that is already opened and in use by a CIFS client.
- When a Unix client device **201** or a PC NFS Windows client device **202** makes an NFS request to remove, rename, or write data to a file **113** that is oplocked by a CIFS client, the file server **110** will enforce CIFS oplock semantics for the file **113**. The file server **110** sends an oplock-break message **140** to the client device **130** holding the oplock, and receives a response from the client device **130**. If the client device **130** closes the file **113**, the NFS request can proceed and the file server **110** allows it to.
- When a Unix client device **201** or a PC NFS Windows client device **202** makes an NFS request to read data from a file **113** that is oplocked by a CIFS client, the file server **110** will enforce CIFS oplock semantics for the file **113**. The file server **110** sends an oplock-break message **140** to the client device **130** holding the oplock, and receives a response from the client device **130**. When the client device **130** either closes the file **113** or flushes its write cache to the file server **110**, the NFS request can proceed and the file server **110** allows it to.
- The file server **110** tests for mutual compatibility for file-open requests from CIFS Windows client devices **203**, and NLM file lock requests from PC NFS Windows client devices **202**, with regard to their specified lock modes. The phrase NLM "file lock" is used herein in place of the known phrase NLM "share lock," further described in the following document: "X/Open CAE Specification: Protocols for X/Open Interworking: XNFS, Issue 4 (X/Open Document Number C218), hereby incorporated by reference as if fully set forth herein. The specified lock mode is determined by the file server **110** by combining the requested access-mode and deny-mode.

To provide these effects, the file server **110** performs the following lock management operations:

- Upon receiving a CIFS file-open request message **140**, the file server **110** tests the file-open request for conflict with existing CIFS and NLM file locks, and for conflict with existing NLM byte-range locks. For the purpose of comparison with newly requested file locks, the file server **110** treats existing NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks and access-mode read-write for exclusive locks.
- Upon receiving a CIFS byte-range lock request message **140**, the file server **110** tests the byte-range lock request for conflict with existing CIFS and NLM byte-range locks.
- Upon receiving an NLM byte-range lock request message **140**, the file server **110** tests the byte-range lock request for conflict with existing CIFS and NLM byte-range locks, and for conflict with existing CIFS file locks.
- Upon receiving an NLM file lock request message **140** from a PC NFS client device **130** (used to simulate a file-open request message **140**), the file server **110** tests the NLM file lock request for conflict with existing CIFS and NLM file locks, and for conflict with existing NLM byte-range locks. For the purpose of comparison with newly requested NLM file locks, the file server **110** treats existing NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks and access-mode read-write for exclusive locks.

Method of Operation (Multi-protocol File Server)

FIG. **3** shows a process flow diagram of a method of operating a multi-protocol file server.

A method **300** of operating a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **310**, the file server **110** is ready to receive the file server request message **140**.

At a step **311**, the file server **110** receives and parses the file server request message **140**. The file server **110** determines if the file server request message **140** uses the NFS file server protocol, the NLM file locking protocol, or the CIFS file server protocol. If the file server request message **140** uses the NFS file server protocol or the NLM file locking protocol, the method **300** continues with the step **312**. If the file server request message **140** uses the CIFS file server protocol, the method **300** continues with the step **313**.

At a step **312**, the file server **110** determines if the request message **140** includes an NFS file server request to perform a file system operation (such as to read or write data, or to modify a directory). Alternatively, the file server **110** determines if the request message **140** includes an NLM file-locking request to obtain an NLM byte-range lock. In either case, the method **300** continues at the flow point **320**.

At a step **313**, the file server **110** determines if the file server request message **140** is to perform a CIFS read or write operation, to obtain a CIFS byte-range lock, or to perform a CIFS file-open operation. In the file server request message **140** is to obtain a CIFS byte-range lock or to perform a CIFS file-open operation, the method **300** con-

tinues at the flow-point **320**. If the file server request message **140** is to perform a CIFS read or write operation, the method continues at the flow-point **330**. If the file server request message **140** is a CIFS "change-notify" request, the method continues at the flow point **350** (the change-notify request is further described with regard to FIG. **6**).

At a flow point **320**, the file server **110** is ready to compare the operation requested by the file server request message **140** with the file-locking status of the file **113**. The file-locking status of the file **113** includes existing file locks and byte-range locks for the file **113**.

At a step **321**, the file server **110** determines the file **113** that is the subject of the file server request message **140**, and determines if the file **113** is oplocked. If the file **113** is oplocked, the method **300** continues with the step **322**. If the file **113** is not oplocked, the method **300** continues with the step **323**.

At a step **322**, the file server **110** breaks the oplock, as described herein. Performance of the step **322** is further described with regard to FIG. **5**. Breaking the oplock can cause the file-locking status of the file **113** to change.

At a step **323**, the file server **110** compares the requested operation with the file-locking status of the file **113**, using a uniform file-locking semantics. In this step, the requested operation can be an NFS read or write operation, an NFS or CIFS directory modification operation, an attempt to obtain an NLM file lock or byte-range lock, or a CIFS file-open operation. Performance of the step **323** and the uniform file-locking semantics are further described with regard to FIG. **4**. If the comparison shows that the requested operation is allowable, the method **300** continues with the step **324**. If the requested operation is not allowable, the method **300** continues with the step **325**.

At a step **324**, the file server **110** performs the requested operation. The method **300** continues at the flow point **340**.

At a step **325**, the file server **110** refuses to perform the requested operation and responds to the client device **130** with an error message. The method **300** continues at the flow point **340**.

At a flow point **330**, the file server **110** is ready to compare the operation requested by the file server request message **140** with the file-locking status of the file **113**.

At a flow point **350**, the file server **110** is ready to perform the change-notify operation, as described herein.

At a step **351**, a first CIFS client device **130** requests a file lock for a directory (using a file system request message **140** to open the directory), and converts the file lock for the directory to a change-monitoring lock on the directory. Performance of this step **351** is further described with regard to FIG. **6**.

At a flow point **340**, the file server **110** has responded to the file server request message **140**, and the method **300** is complete with regard to that file server request message **140**.

Method of Operation (Cross-protocol Lock Manager)

FIG. **4** shows a process flow diagram of a method of operating a cross-protocol lock manager in a multi-protocol file server.

A method **400** of operating a cross-protocol lock manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **410**, the file server **110** is ready to compare the requested operation in the file server request message **140**, with the file-locking status of the file **113**.

The file server **110** uses a uniform file-locking semantics, so as to model file-locking aspects of any requested operation from any file server protocol in the same way. The uniform file-locking semantics identifies a uniform set of file locks, each including an access-mode for the requesting client device **130** and a deny-mode for all other client devices **130**.

In a preferred embodiment, the access-mode can be one of three possibilities—read-only, write-only, or read-write. Similarly, in a preferred embodiment, the deny-mode can be one of four possibilities—deny-none, deny-read, deny-write, or deny-all.

After a first client device **130** obtains a file lock for a file **113**, a second client device **130** can only access that file **113** if the lock mode determined by the file server **110** to be requested by the second client device **130** is compatible with the file- locking status of the file **113**. For example, a first client device **130** can obtain a file lock for a file **113** with a deny-mode deny-write. A second NFS client device **130** could attempt to write to the file **113**, or a second CIFS client device **130** could attempt to open the file **113** with an access-mode including write access. In either such case (if the file lock for the file **113** is not an opportunistic lock, as further described herein), the file server **110** will deny the request by the second client device **130**.

As noted herein, the file server **110** performs the comparison of the file lock with the access requested by the second client device **130** at differing times, in response to the file server protocol used by the second client device **130**:

- If the second client device **130** uses the CIFS file server protocol to open the file **113**, the file server **110** checks the file-locking status of the file **113** at file-open time.
- If the second client device **130** uses the NFS file server protocol to read or write to the file **113**, the file server **110** checks the file-locking status of the file **113** at the time of the actual file system operation. This also applies to file system operations that have the effect of removing the file from view of the first client device **130**, such as operations to move, remove, or rename the file **113**.
- If the second client device **130** uses the CIFS file server protocol to request a byte-range lock, the file server **110** checks the file-locking status of the file **113** for conflict with other CIFS or NLM byte-range locks, at the time the byte-range lock is requested. The file server **110** does not check for conflict with other CIFS file locks at the time the byte-range lock is requested, because those were checked at file-open time.
- If the second client device **130** uses the NLM protocol to request a byte-range lock, the file server **110** checks the file-locking status of the file **113** for conflict with existing CIFS or NLM byte-range locks, and for conflict with existing CIFS file locks, at the time the byte-range lock is requested.

At a step **421**, the file server **110** determines if there is already more than one file lock associated with the file **113**. If so, the method **400** continues with the step **422**. If not, the method continues with the step **411**.

At a step **422**, the file server **110** combines file locks already associated with the file **113** into a single equivalent file lock associated with the file **113**. To perform this step **422**, the file server **110** cross-indexes in table 1 a cumulative file lock with each pre-existing file lock, until all pre-existing file locks have been cumulated together.

Table 1 shows a lock conversion table in a multi-protocol file server with unified file-locking semantics.

TABLE 1

		Existing file lock mode										
		NULL	A: R D: DN	A: W D: DN	A: RW D: DN	A: R D: DR	A: W D: DR	A: RW D: DR	A: R D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
New lock mode	NULL	NULL	A: R D: DN	A: W D: DN	A: RW D: DN	A: R D: DR	A: W D: DR	A: RW D: DR	A: R D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
	A: R D: DN	A: R D: DN	A: R D: DN	A: RW D: DN	A: RW D: DN	A: R D: DR	A: RW D: DR	A: RW D: DR	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
	A: W D: DN	A: W D: DN	A: RW D: DN	A: W D: DN	A: RW D: DN	A: RW D: DR	A: W D: DR	A: RW D: DR	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
	A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DN	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
	A: R D: DR	A: R D: DR	A: R D: DR	A: RW D: DR	A: RW D: DR	A: R D: DR	A: RW D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
	A: W D: DR	A: W D: DR	A: RW D: DR	A: W D: DR	A: RW D: DR	A: RW D: DR	A: W D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: RW D: DR	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA
	A: R D: DW	A: R D: DW	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: R D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
	A: W D: DW	A: W D: DW	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: RW D: DW	A: W D: DW	A: RW D: DW	A: Any D: DA
	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: RW D: DW	A: RW D: DW	A: RW D: DW	A: Any D: DA
	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA	A: Any D: DA

Lock Conversion Matrix
A = Access Mode (R = Read, W = Write, RW = Read-Write, Any = any one of R or W or RW)
D = Deny Mode (DN = Deny None, DR = Deny Read, DW = Deny Write, DA = Deny All)

At a step **411**, the file server **110** determines the nature of the requested operation in the file server request message **140**. If the requested operation is a CIFS file-open operation, the method **400** continues with the step **423**. If the requested operation is an NFS file server operation, the method **400** continues with the step **431**. If the requested operation is either a CIFS request an NLM request for a byte-range lock, the file system **110** continues with the step **441**.

At a step **423**, the file server **110** compares the file lock already associated with the file **113** with the file open requested by the second client device **130**. To perform this step **423**, the file server **110** cross-indexes in table 2 the pre-existing file lock and the requested new access-mode and deny-mode, and allows or denies the requested new access-mode and deny-mode in response to the associated table entry.

If the file server **110** allows the requested new access-mode and deny-mode, the method **400** performs the step **424**. If the file server **110** denies the requested new access-mode and deny-mode, the method **400** does not perform the step **424**.

Table 2 shows a cross-index of attempted file locks in a multi-protocol file server with unified file-locking semantics.

TABLE 2

		Pre-existing file lock										
		NULL	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
New mode being requested	NULL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	A: R D: DN	✓	✓	X	✓	✓	X	✓	✓	X	✓	X
	A: R D: DR	✓	X	X	X	✓	X	✓	✓	X	✓	X
	A: R D: DW	✓	✓	X	✓	X	X	X	X	X	X	X
	A: W D: DN	✓	✓	✓	X	✓	✓	X	✓	✓	X	X
	A: W D: DR	✓	X	X	X	✓	✓	X	X	X	X	X
	A: W D: DW	✓	✓	✓	X	X	X	X	X	X	X	X
	A: RW D: DN	✓	✓	X	X	✓	X	X	✓	X	X	X
	A: RW D: DR	✓	X	X	X	✓	X	X	X	X	X	X
	A: RW D: DW	✓	✓	X	X	X	X	X	X	X	X	X

TABLE 2-continued

	Pre-existing file lock										
	NULL	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
A: Any D: DA	✓	X	X	X	X	X	X	X	X	X	X

Multi-Protocol Lock Compatibility Matrix
A = Access Mode,
D = Deny Mode
✓ = New request will be granted.
X = New request will be denied.

As shown in table 2, each pair of pre-existing file lock and requested new access-mode and deny-mode has an associated decision to allow or to deny the requested new access-mode and deny-mode.

If the file server **110** is checking for conflicts between an existing CIFS file lock and a new request to perform a file-open operation, the existing CIFS file lock is cross-indexed against the access-mode and deny-mode requested in the new file-open request.

If the file server **110** is checking for conflicts between existing file locks and a new NFS request to perform a file read or write operation, the aggregate lock mode (the combination of existing file locks) is cross-indexed against the access-mode required to perform the new request.

If the file server **110** is checking for conflicts between existing file locks or byte-range locks, and a new request for a NLM byte-range lock, the existing file locks and byte-range locks are cross-indexed against a lock mode equivalent to the new NLM byte-range lock request. For the purpose of comparing with existing file locks, the file server **110** treats newly requested NLM byte-range locks as having deny-mode deny-none, and as having access-mode read-only for nonexclusive locks (also called "read locks") and access-mode read-write for exclusive locks (also called "write locks"). For the purpose of comparing with existing byte-range locks, the file server **110** treats newly requested NLM byte-range locks as having access-mode read-only and deny-mode deny-write for read locks, and as having access-mode read-write and deny-mode deny-all for write locks.

The method **400** then continues at the flow point **450**.

At a step **431**, the file server **110** compares the file-locking status of the file **113** with the operation requested by the second client device **130**. To perform this step **431**, the file server **110** compares the deny-mode for the file lock with the requested operation, and allows or denies the requested operation in response thereto.

The method **400** then continues at the flow point **450**.

At a step **441**, the file server **110** compares the file-locking status of the file **113** with the NLM byte-range lock requested by the second client device **130**. In a preferred embodiment, CIFS byte-range lock requests are only checked against byte-range locks because they require a prior CIFS file open operation at which existing file locks were already checked. To perform this step **441**, the file server **110** cross-indexes in table 3 the pre-existing file-locking status and the requested byte-range lock, and allows or denies the requested byte-range lock in response to the associated table entry.

If the file server **110** allows the requested new NLM byte-range lock, the method **400** performs the step **442**. If the file server **110** denies the requested new byte-range lock, the method **400** does not perform the step **442**.

Table 3 shows a cross-index of existing file locks and newly requested NLM byte-range locks in a multi-protocol file server with unified file-locking semantics.

TABLE 3

	Existing lock mode										
	None	A: R D: DN	A: R D: DR	A: R D: DW	A: W D: DN	A: W D: DR	A: W D: DW	A: RW D: DN	A: RW D: DR	A: RW D: DW	A: Any D: DA
Write Lock	✓	✓	✓	X	✓	✓	X	✓	✓	X	X
Read Lock	✓	✓	X	✓	✓	X	✓	✓	X	✓	X

Compatibility of new NLM byte-range locks with existing file locks
A = Access Mode,
D = Deny mode
✓ = New NLM byte-range lock request will be granted.
X = New NLM byte-range lock request will be denied.

As shown in table 3, each pair of existing file lock and newly requested NLM byte-range lock has an associated decision to allow or to deny the requested new byte-range lock.

At a step **442**, the file server **110** associates the requested new byte-range lock with the file **113** as a new additional byte-range lock.

The method **400** then continues at the flow point **450**.

At a flow point **450**, the file server **110** has compared the requested operation in the file server request message **140**

with the file-locking status of the file **113**, and allowed or denied the requested operation.

Method of Operation (Oplock Manager)

FIG. **5** shows a process flow diagram of a method of operating an oplock manager in a multi-protocol file server.

A method **500** of operating a oplock manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

Oplocks are known in the art of file-locking in Windows operating system environments. They are further described in documentation available for the "Windows NT 4.0" operating system, available from Microsoft Corporation of Redmond, Wash., including for example the CIFS IETF specification, available via the FTP protocol at host fip.microsoft.com, directory/developr/drg/CIFS, files cifs6.doc or cifs6.txt, hereby incorporated by reference as if filly set forth herein.

At a flow point **510**, the file server **110** is ready to receive a request from a CIFS first client device **130** to open a file **113**.

At a step **511**, the file server **110** receives a file-open request for a file **113** from a CIFS first client device **130**. The file-open request designates an access-mode and a deny-mode.

At a step **512**, the file server **110** determines that it should allow the request, and grants the first client device **130** a file lock with the designated access-mode and deny-mode.

At a step **513**, if the client device **130** has requested an oplock on the file open request, the file server **110** grants the first client device **130** an oplock at a level of exclusivity possibly greater than the first client device **130** actually requires.

For example, when a CIFS first client device **130** opens a file **113** with the access-mode read-only and deny-mode deny-write, the file server **110** associates a file lock of that type with the file **113**. The file server **110** further associates an oplock with the file **113** with the access-mode read-write and deny-mode deny-all.

At a flow point **520**, the file server **110** has responded to the request from the CIFS first client device **130** for a file lock for a file **113**.

At a flow point **530**, a second client device **130** attempts to open the file **113**.

At a step **531**, the file server **110** receives either a file-open request from a second CIFS client device **130** or a NLM file lock request from a PC NFS client device **130**.

As part of performing this step **531**, the file server **110** suspends execution of the request by the second client device **130** while it breaks the oplock and obtains a response from the holder of the oplock, the first client device **130**.

At a step **532**, the file server **110** breaks the oplock by sending an "oplock-break" message **140** to the CIFS first client device **130**.

When the second client device **130** is a CIFS client device **130**, this is already expected. When the second client device **130** is an NFS client device **130**, the file server **110** delays its response to the NFS (or NLM) protocol request message **140** until the CIFS first client device **130** responds to the "oplock-break" message **140**.

At a step **533**, the CIFS first client device **130** receives the "oplock-break" message **140**, and can respond to the message **140** in one of two ways:

- The CIFS first client device **130** can close the file **113** (thus removing the file lock associated with the file-open); or
- The CIFS first client device **130** can flush all outstanding CIFS write and byte-range lock requests for the file **113** that are being cached locally at the client device **130** (that is, it can forward the results of those file system operations to the file server **110**), and discard any read-ahead data it has obtained for the file **113**. Read-ahead data should be discarded because the second client device **130** might subsequently write new data to the file, invalidating the read-ahead data.

At a step **534**, the file server **110** receives the response from the CIFS first client device **130**.

At a step **535**, the file server **110** determines if the CIFS first client device **130** has maintained the file **113** open, and if so, compares the lock mode implied by the request by the second client device **130** against the new file-locking status of the file **113**. If the file server **110** determines that the request by the second client device **130** is allowed to proceed, it continues with the flow point **540**. If the file server **110** determines that the request by the second client device **130** is not allowed to proceed, it denies the request.

At a flow point **540**, the file server **110** is ready to proceed to allow the request from the second client device **130** noted in the step **531**.

Method of Operation (Change-notify Manager)

FIG. **6** shows a process flow diagram of a method of operating a change-notify manager in a multi-protocol file server.

A method **600** of operating a change-notify manager in a multi-protocol file server includes a set of process steps and flow points as described herein, to be performed by the file server **110** in cooperation with at least one client device **130**.

At a flow point **610**, the file server **110** is ready to receive the file server request message **140**.

At a step **611**, the file server **110** receives a file-open request message **140** from a first CIFS client device **130**, designating a directory on the file server **110**. The file server **110** determines that it should allow the file-open request and grants a CIFS file lock on the directory to the first CIFS client device **130**.

At a step **612**, the file server **110** receives a change-notify request message from the first CIFS client device **130**, referencing the open directory, to convert the file lock on the open directory to a change-monitoring lock.

At a step **613**, the file server **110** converts the file lock on the open directory to a change-monitoring lock on the designated directory.

At a flow point **620**, the "change-monitoring" lock has been associated with the designated directory, and the first CIFS client device **130** is ready to be notified of changes to that directory.

At a step **621**, the file server **110** receives a file server request message **140** from a second client device **130**, requesting a change to the designated directory, and thus triggering a change notification to the first client device **130**. (Types of change include file creation, file deletion, file rename, file move between directories, file attribute change, and file modification time change.) The file server request message **140** from the second client device **130** can be either CIFS or NFS. The second client device **130** can be any one of a Unix NFS client device **201**, a PC NFS client device **202**, or a CIFS Windows client device **203**.

At a step **622**, the file server **110** notifies the first client device **130**, which holds the "change-monitoring" lock, of the changes noted in the step **621**, containing possibly multiple entries, each of which specifies both the name of the changed file **113** or subdirectory within the monitored

directory and the type of change. If there is more than one such first client device **130**, the file server **110** notifies all of them.

Change-notification is known in the art of file-locking in Windows NT operating system environments. It is further described in documentation available for the "Windows NT 4.0" operating system, available from Microsoft Corporation of Redmond, Wash., including for example the CIFS IETF specification, available via the FTP protocol at host ftp.microsoft.com, directory/developr/drg/CIFS, files cifs6.doc or cifs6.txt, hereby incorporated by reference as if fully set forth herein.

At a flow point **630**, the file server **110** has notified the first CIFS client device **130** of changes to the designated directory, and is ready for a next message **140**.

Alternative Embodiments

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

Technical Appendix

Other and further information about the invention is included in a technical appendix enclosed with this application. This technical appendix includes 30 pages (including drawings) and is hereby incorporated by reference as if fully set forth herein.

The invention claimed is:

1. A method of enforcing uniform locking semantics among a set of client devices that use a plurality of diverse locking protocols, comprising the steps of:
- granting an opportunistic lock on a selected data set to a first client device in response to a first message using a first protocol; and
- breaking said opportunistic lock in response to a second message using a second protocol;
- wherein the first protocol and the second protocol are different ones of said plurality of diverse locking protocols.

2. A method as in claim **1**, wherein said first protocol is CIFS.

3. A method as in claim **2**, wherein said second protocol is NFS or NLM.

4. A method as in claim **1**, wherein said step of breaking further comprises steps of:
- sending an oplock-break message to said first client device in response to said second message;
- delaying execution of a file system request indicated by said second message;
- receiving a response to said oplock-break message from said first client device; and
- processing and responding to said second message after said step of receiving.

5. A method as in claim **1**, wherein said uniform locking semantics include:
- a change-monitoring lock type capable of being requested by said first client device using said first protocol; and
- a change notification being triggered by a second client device using said second protocol different from said first protocol.

6. A method as in claim **5**, wherein said first protocol is CIFS.

7. A method as in claim **6**, wherein said second protocol is NFS.

8. A method as in claim **1**, wherein said locking semantics include a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

9. A method as in claim **1**, wherein said locking semantics include:
- a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and
- a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;
- wherein said method further comprises a comparison of said first lock mode with said second lock mode.

10. A method as in claim **9**, wherein said comparison includes a lock compatibility matrix.

11. A method as in claim **9**, wherein said comparison includes a lock conversion matrix.

12. A method as in claim **9**, wherein said second lock mode is responsive to a request for a byte-range lock.

13. A method as in claim **9**, wherein said second lock mode is responsive to a request for a NLM file lock.

14. A method of enforcing uniform file-locking semantics among a set of client devices that use a plurality of diverse file locking protocols, comprising the steps of:
- receiving a first message using a first protocol, said first message being operative to lock at least a portion of a selected file;
- granting an opportunistic lock on said at least a portion of a selected file, in response to said first message;
- receiving a second message using a second protocol different from the first protocol, said second message being operative to request access to said at least a portion of the selected file; and
- comparing said access requested by said second message with said lock, and
- denying said access if prohibited by said lock, and otherwise,
- breaking said opportunistic lock in response to said second message.

15. A method as in claim **14**, wherein said first protocol is CIFS.

16. A method as in claim **15**, wherein said second protocol is NLM.

17. A method as in claim **15**, wherein said second protocol is NFS.

18. A method as in claim **14**,
- wherein said step of receiving said second message includes the step of recognizing said second message as being for obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and
- wherein said step of comparing includes the step of testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file locking protocols.

19. A method as in claim **18**, wherein said step of testing is responsive to one of said diverse file locking protocols used for said second message.

20. A method as in claim **18**, wherein said step of testing operates at file-open time for said first protocol and at an access time for said second protocol.

21. A method as in claim **18**, wherein said step of testing operates at file-open time for said first protocol and at a lock-request time for said second protocol.

22. A method as in claim **14**,

wherein said step of receiving said second message includes the step of recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

wherein said step of comparing includes the step of testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

23. A method as in claim **14**,

wherein said step of receiving said second message includes the step of recognizing said second message for reading from or writing to a file using said second protocol; and

wherein said step of comparing includes the step of testing whether accessing said file as attempted by said second message would conflict with existing locks created by messages using the same or other diverse file locking protocols.

24. A method as in claim **14**,

wherein said step of receiving said first message includes the step of granting said opportunistic lock in response to said first message; and

wherein said step of comparing includes the step of breaking said opportunistic lock in response to said second message.

25. A method as in claim **24**, wherein said step of breaking includes the steps of:

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

26. A method as in claim **25**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

27. A storage server comprising:

mass storage;

an interface to a network; and

a processor, the storage server operating under control of the processor to enforce uniform locking semantics for accessing the mass storage across the network, said semantics enforced among a set of client devices that use a plurality of diverse locking protocols, said processor performing steps including the steps of: (a) granting an opportunistic lock on a selected data set to a first client device in response to a first message using a first protocol, and (b) breaking said opportunistic lock in response to a second message using a second protocol, wherein the first protocol and the second protocol are different ones of said plurality of diverse locking protocols.

28. A storage server as in claim **27**, wherein said first protocol is CIFS.

29. A storage server as in claim **28**, wherein said second protocol is NFS or NLM.

30. A storage server as in claim **27**, wherein said step of breaking further comprises steps of:

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

31. A storage server as in claim **27**, wherein said uniform locking semantics include:

a change-monitoring lock type capable of being requested by said first client device using said first protocol; and

a change notification being triggered by a second client device using said second protocol different from said first protocol.

32. A storage server as in claim **31**, wherein said first protocol is CIFS.

33. A storage server as in claim **32**, wherein said second protocol is NFS.

34. A storage server as in claim **27**, wherein said locking semantics include a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

35. A storage server as in claim **27**, wherein said locking semantics include:

a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and

a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;

wherein said storage server is responsive to comparison of said first lock mode with said second lock mode.

36. A storage server as in claim **35**, wherein said comparison includes a lock compatibility matrix.

37. A storage server as in claim **35**, wherein said comparison includes a lock conversion matrix.

38. A storage server as in claim **35**, wherein said second lock mode is responsive to a request for a byte-range lock.

39. A storage server as in claim **35**, wherein said second lock mode is responsive to a request for a NLM file lock.

40. A file server comprising:

mass storage;

an interface to a network; and

a processor, the file server operating under control of the processor to enforce uniform file-locking semantics for accessing the mass storage across the network, said semantics enforced among a set of client devices that use a plurality of diverse file locking protocols, said processor performing steps including the steps of: receiving a first message using a first protocol, said first message being operative to lock at least a portion of a selected file, granting an opportunistic lock on said at least a portion of a selected file, in response to said first message, receiving a second message using a second protocol different from the first protocol, said second message being operative to request access to said at least a portion of the selected file, comparing said access requested by said second message with said lock, and denying said access if prohibited by said lock, and otherwise, breaking said opportunistic lock in response to said second message.

41. A file server as in claim **40**, wherein said first protocol is CIFS.

42. A file server as in claim **41**, wherein said second protocol NLM.

43. A file server as in claim **41**, wherein said second protocol is NFS.

44. A file server as in claim **40**,

wherein said step of receiving said second message includes the step of recognizing said second message as

being for obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and

wherein said step of comparing includes the step of testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file locking protocols.

45. A file server as in claim **44**, wherein said step of testing is responsive to one of said diverse file locking protocols used for said second message.

46. A file server as in claim **44**, wherein said step of testing operates at file-open time for said first protocol and at an access time for said second protocol.

47. A file server as in claim **44**, wherein said step of testing operates at file-open time for said first protocol and at a lock-request time for said second protocol.

48. A file server as in claim **40**,

wherein said step of receiving said second message includes the step of recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

wherein said step of comparing includes the step of testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

49. A file server as in claim **40**,

wherein said step of receiving said second message includes the step of recognizing said second message for reading from or writing to a file using said second protocol; and

wherein said step of comparing includes the step of testing whether accessing said file as attempted by said second message would conflict with existing locks created by messages using the same or other diverse file locking protocols.

50. A file server as in claim **40**,

wherein said step of receiving said first message includes the step of granting said opportunistic lock in response to said first message; and

wherein said step of comparing includes the step of breaking said opportunistic lock in response to said second message.

51. A file server as in claim **50**, wherein said step of breaking includes the steps of:

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

52. A file server as in claim **51**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

53. A memory storing information including instructions, the instructions executable by a processor to enforce uniform locking semantics, said semantics enforced among a set of client devices that use a plurality of diverse locking protocols, said instructions comprising the steps of:

granting an opportunistic lock on a selected data set to a first client device in response to a first message using a first protocol; and

breaking said opportunistic lock in response to a second message using a second protocol, wherein the first protocol and the second protocol are different ones of said plurality of diverse locking protocols.

54. A memory as in claim **53**, wherein said first protocol is CIFS.

55. A memory as in claim **54**, wherein said second protocol is NFS or NLM.

56. A memory as in claim **53**, wherein said step of breaking further comprises steps of:

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

57. A memory as in claim **53**, wherein said uniform locking semantics include:

a change-monitoring lock type capable of being requested by said first client device using said first protocol; and

a change notification being triggered by a second client device using said second protocol different from said first protocol.

58. A memory as in claim **57**, wherein said first protocol is CIFS.

59. A memory as in claim **58**, wherein said second protocol is NFS.

60. A memory as in claim **53**, wherein said locking semantics include a lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol.

61. A memory as in claim **53**, wherein said locking semantics include:

a first lock mode determined in response to an access-mode and a deny-mode requested by said first client device using said first protocol; and

a second lock mode determined in response to a message from a second client device using said second protocol different from said first protocol;

a comparison of said first lock mode with said second lock mode.

62. A memory as in claim **61**, wherein said comparison includes a lock compatibility matrix.

63. A memory as in claim **61**, wherein said comparison includes a lock conversion matrix.

64. A memory as in claim **61**, wherein said second lock mode is responsive to a request for a byte-range lock.

65. A memory as in claim **61**, wherein said second lock mode is responsive to a request for a NLM file lock.

66. A memory storing information including instructions, the instructions executable by a processor to enforce uniform file-locking semantics, said semantics enforced among a set of client devices that use a plurality of diverse file locking protocols, said instructions comprising the steps of:

receiving a first message using a first protocol, said first message being operative to lock at least a portion of a selected file;

granting an opportunistic lock on said at least a portion of a selected file, in response to said first message;

receiving a second message using a second protocol different from the first protocol, said second message being operative to request access to said at least a portion of the selected file; and

comparing said access requested by said second message with said lock, and

denying said access if prohibited by said lock, and otherwise,

breaking said opportunistic lock in response to said second message.

67. A memory as in claim **66**, wherein said first protocol is CIFS.

68. A memory as in claim **67**, wherein said second protocol is NLM.

69. A memory as in claim **67**, wherein said second protocol is NFS.

70. A memory as in claim **66**,

wherein said step of receiving said second message includes the step of recognizing said second message as being for obtaining a byte-range lock on a file using said second protocol, said byte-range lock having a lock type; and

wherein said step of comparing includes the step of testing whether obtaining said byte-range lock having said lock type would conflict with existing locks created by messages using the same or other diverse file locking protocols.

71. A memory as in claim **70**, wherein said step of testing is responsive to one of said diverse file locking protocols used for said second message.

72. A memory as in claim **70**, wherein said step of testing operates at file-open time for said first protocol and at an access time for said second protocol.

73. A memory as in claim **70**, wherein said step of testing operates at file-open time for said first protocol and at a lock-request time for said second protocol.

74. A memory as in claim **66**,

wherein said step of receiving said second message includes the step of recognizing said second message for opening a file using said second protocol, said second message including a requested access-mode; and

wherein said step of comparing includes the step of testing whether accessing said file using said requested access-mode would conflict with existing locks created by messages using the same or other diverse file locking protocols.

75. A memory as in claim **66**,

wherein said step of receiving said second message includes the step of recognizing said second message for reading from or writing to a file using said second protocol; and

wherein said step of comparing includes the step of testing whether accessing said file as attempted by said second message would conflict with existing locks created by messages using the same or other diverse file locking protocols.

76. A memory as in claim **66**,

wherein said step of receiving said first message includes the step of granting said opportunistic lock in response to said first message; and

wherein said step of comparing includes the step of breaking said opportunistic lock in response to said second message.

77. A memory as in claim **76**, wherein said step of breaking includes the steps of:

sending an oplock-break message to said first client device in response to said second message;

delaying execution of a file system request indicated by said second message;

receiving a response to said oplock-break message from said first client device; and

processing and responding to said second message after said step of receiving.

78. A memory as in claim **77**, wherein said response to said oplock-break message includes an oplock-break acknowledgement message or a file close message.

79. A file server comprising:

mass storage means for storing files;

interface means for interfacing to a network; and

processing means for controlling the file server to enforce uniform file-locking semantics for accessing the mass storage across the network, said semantics enforced among a set of client devices that use a plurality of diverse file locking protocols, including means for granting an opportunistic lock on a selected file to a first client device in response to a first message using a first protocol; and

means for breaking said opportunistic lock in response to a second message using a second protocol from a second client device, wherein the first protocol and the second protocol are different ones of said plurality of diverse file locking protocols.

* * * * *

Exhibit C

US007293152B1

(12) **United States Patent**
Srinivasan et al.

(10) **Patent No.: US 7,293,152 B1**
(45) **Date of Patent: *Nov. 6, 2007**

(54) **CONSISTENT LOGICAL NAMING OF INITIATOR GROUPS**

(75) Inventors: **Mohan Srinivasan**, Cupertino, CA (US); **Herman Lee**, Mountain View, CA (US)

(73) Assignee: **Network Appliance, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 560 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/421,576**

(22) Filed: **Apr. 23, 2003**

(51) **Int. Cl.**
G06F 12/06 (2006.01)

(52) **U.S. Cl.** **711/202**; 711/4; 711/112; 711/111; 711/203; 709/217

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,163,131	A		11/1992	Row et al.
5,355,453	A		10/1994	Row et al.
5,485,579	A		1/1996	Hitz et al.
5,802,366	A		9/1998	Row et al.
5,819,292	A		10/1998	Hitz et al.
5,897,661	A	*	4/1999	Baranovsky et al. 711/170
5,918,229	A	*	6/1999	Davis et al. 707/10
5,931,918	A		8/1999	Row et al.
5,941,972	A		8/1999	Hoese et al.
5,948,110	A		9/1999	Hitz et al.
5,950,225	A		9/1999	Kleiman
5,963,962	A		10/1999	Hitz et al.
6,038,570	A		3/2000	Hitz et al.
6,065,037	A		5/2000	Hitz et al.
6,081,879	A	*	6/2000	Arnott 711/173
6,105,122	A	*	8/2000	Muller et al. 712/1
6,138,126	A		10/2000	Hitz et al.
6,148,349	A	*	11/2000	Chow et al. 710/33

(Continued)

OTHER PUBLICATIONS

David Hitz et al. TR3002 File System Design for a NFS File Server Appliance published by Network Appliance, Inc., Jul. 9, 2001.

Common Internet File System (CIFS) Version: CIFS-Spec 0.9, Storage Networking Industry Association (SNIA), Draft SNIA CIFS Documentation Work Group Work-in-Progress, Revision Date: Mar. 26, 2001.

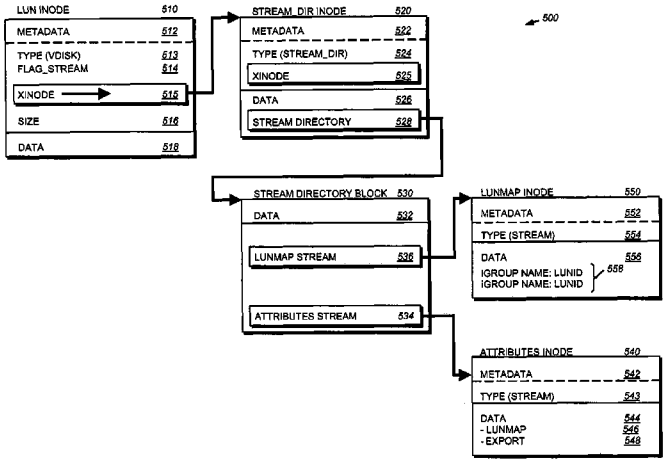
(Continued)

Primary Examiner—B. James Peikari
(74) *Attorney, Agent, or Firm*—Cesari and McKenna LLP

(57) **ABSTRACT**

A technique enables efficient access to logical unit numbers (luns) or virtual disks (vdisks) stored on a storage system, such as a multi-protocol storage appliance. The technique allows a grouping of initiators by a "human friendly" logical name that is mapped to a lun or vdisk on the storage appliance. The initiators are clients operating in, e.g., a storage area network (SAN) environment that initiate requests for the vdisk using block-based access protocols, such as the Small Computer Systems Interface (SCSI) protocol encapsulated over TCP/IP (iSCSI) or over fibre channel (FCP). The technique enables access to the vdisk by all initiators that are members of the initiator group (igroup). An igroup is a logical named entity that is assigned to one or more addresses associated with one or more initiators. These addresses may comprise fibre channel (FC) world wide name (WWN) or iSCSI name identifiers (IDs). Therefore, rather than having to specify these IDs when desiring access to a vdisk, an initiator need only specify the human friendly name of the igroup.

38 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

6,289,356	B1	9/2001	Hitz et al.	
6,425,035	B2	7/2002	Hoese et al.	
6,526,478	B1 *	2/2003	Kirby	711/114
7,055,014	B1 *	5/2006	Pawlowski et al.	711/202
7,076,509	B1 *	7/2006	Chen et al.	707/202
7,107,385	B2 *	9/2006	Rajan et al.	711/4
2002/0112022	A1	8/2002	Kazar et al.	
2002/0116593	A1	8/2002	Kazar et al.	
2002/0156984	A1 *	10/2002	Padovano	711/148
2004/0030668	A1 *	2/2004	Pawlowski et al.	707/1
2004/0054776	A1 *	3/2004	Klotz et al.	709/224
2004/0064458	A1 *	4/2004	Hagarty	707/100
2004/0123063	A1 *	6/2004	Dalal et al.	711/170
2005/0228835	A1 *	10/2005	Roa	707/204
2006/0242179	A1 *	10/2006	Chen et al.	707/100

OTHER PUBLICATIONS

Fielding et al. (1999) Request for Comments (RFC) 2616, HTTP/1.1.

Maintenance Procedures ND (BC) nd-network disk control Feb. 1985.

Misc. Reference Manual Pages ND (4P) nd-network disk driver Jul. 26, 1985.

Asante EN/SC Adapter Family Installation Guide May 1994.

Asante Desktop EN/SC Adapters User's Manual Apr. 1996.

Performance Without Compromise: The Virtual Storage Architecture 1997.

Anthony J. McGregor Department of Computer Science, University of Waikato Dissertation: Block-Based Distributed File Systems Jul. 1997.

* cited by examiner

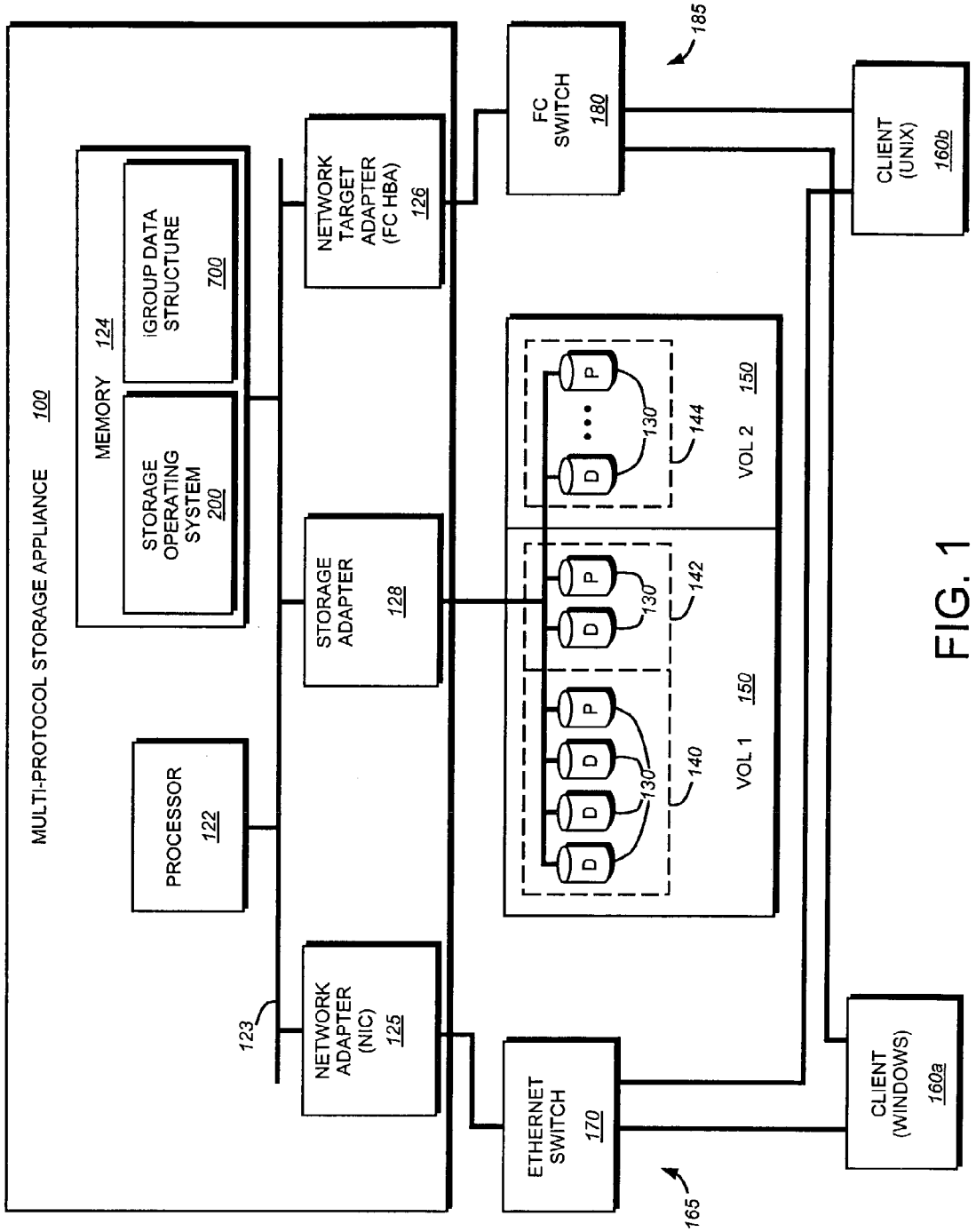

MULTI-PROTOCOL STORAGE APPLIANCE 100
MEMORY 124
iGROUP DATA STRUCTURE 700
STORAGE OPERATING SYSTEM 200
PROCESSOR 122
NETWORK TARGET ADAPTER (FC HBA) 126
STORAGE ADAPTER 128
NETWORK ADAPTER (NIC) 125
123
FC SWITCH 180
185
ETHERNET SWITCH 170
165
CLIENT (UNIX) 160b
CLIENT (WINDOWS) 160a
VOL 1 150
VOL 2 150
D
P
130
140
142
144

FIG. 1

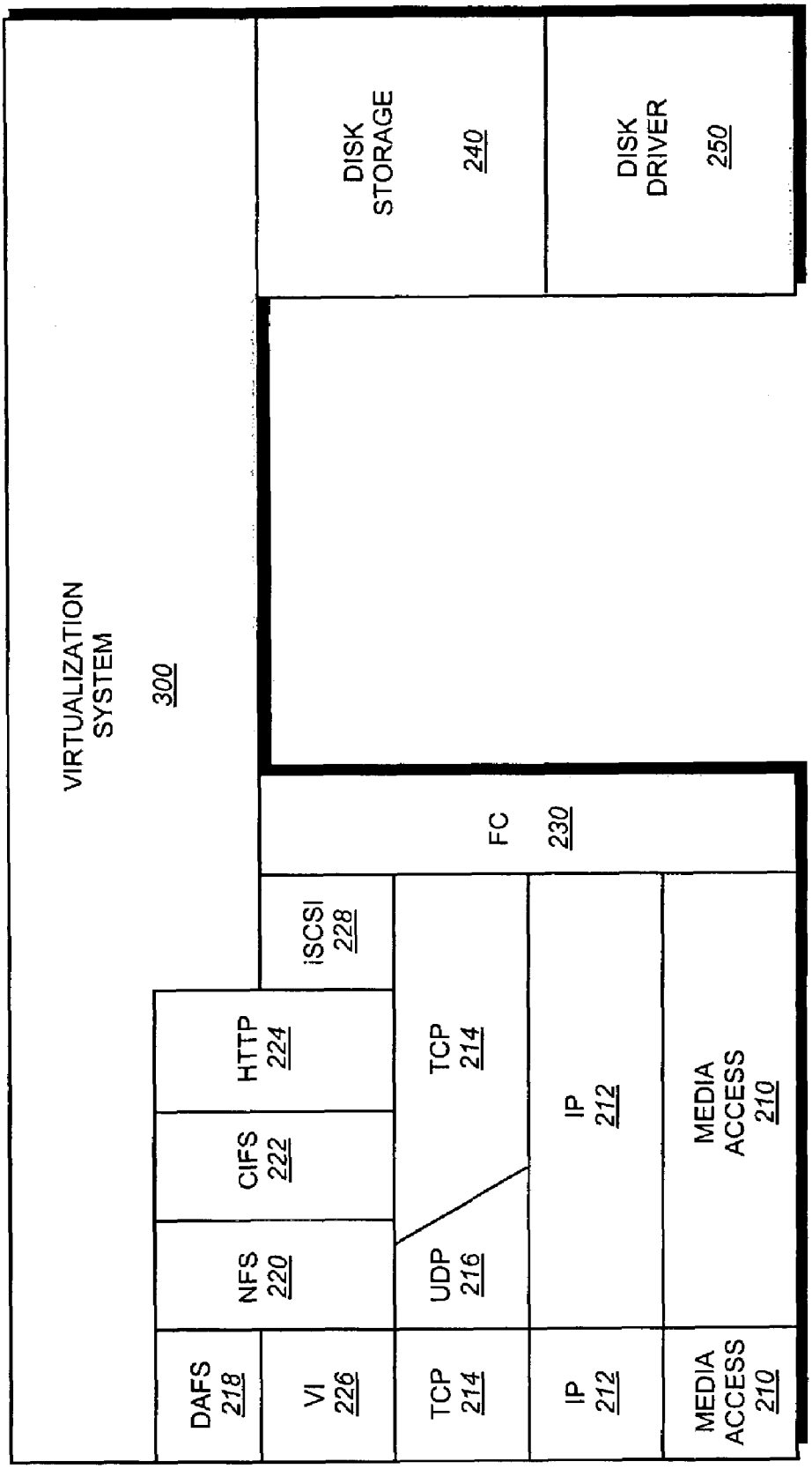

VIRTUALIZATION SYSTEM 300
DISK STORAGE 240
DISK DRIVER 250
FC 230
iSCSI 228
HTTP 224
CIFS 222
NFS 220
DAFS 218
VI 226
TCP 214
UDP 216
TCP 214
IP 212
IP 212
MEDIA ACCESS 210
MEDIA ACCESS 210

FIG. 2

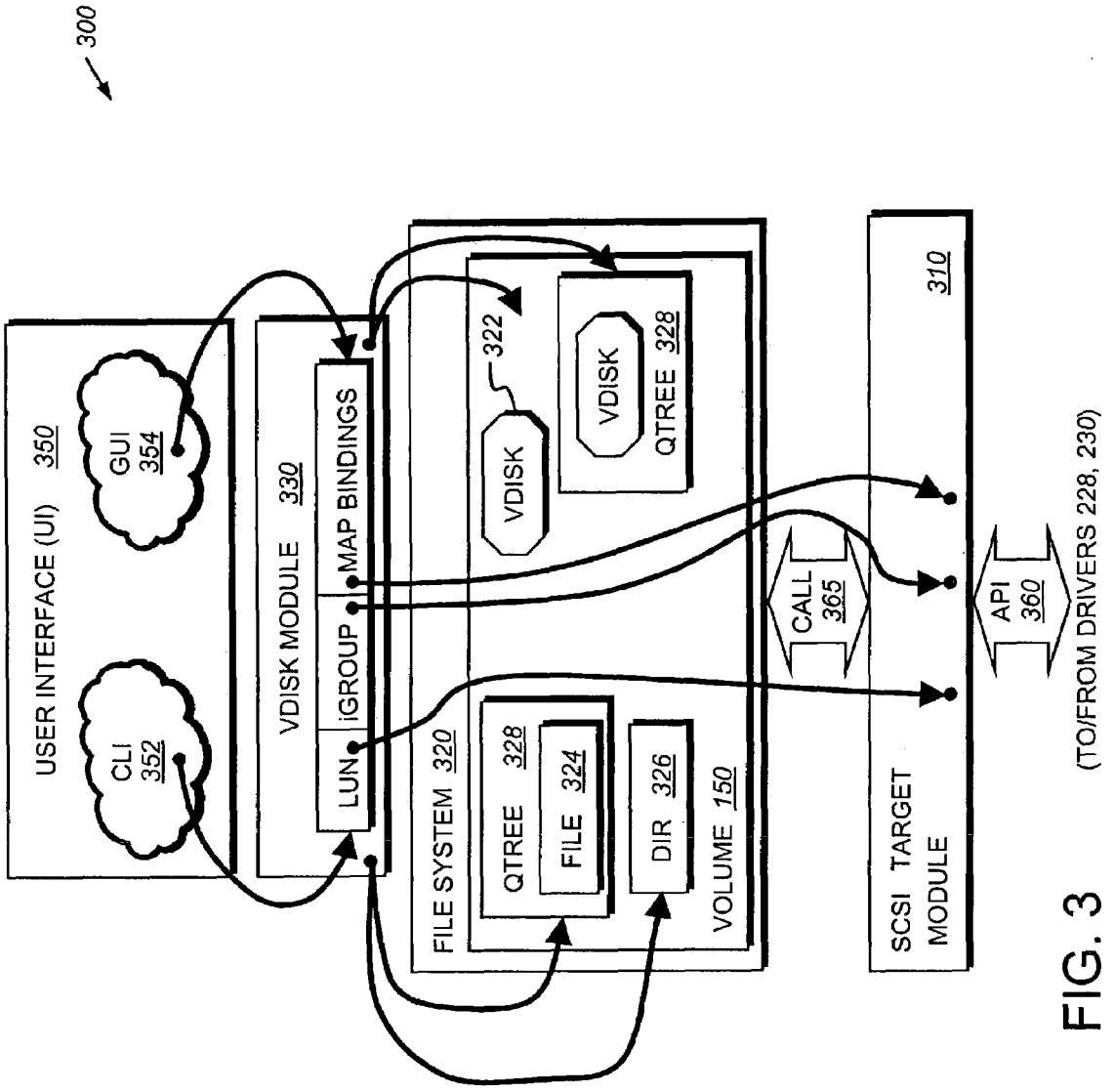

300
USER INTERFACE (UI) 350
GUI 354
CLI 352
VDISK MODULE 330
MAP BINDINGS
iGROUP
LUN
322
VDISK
VDISK
QTREE 328
FILE SYSTEM 320
QTREE 328
FILE 324
DIR 326
VOLUME 150
CALL 365
API 360
310
SCSI TARGET MODULE
(TO/FROM DRIVERS 228, 230)

FIG. 3

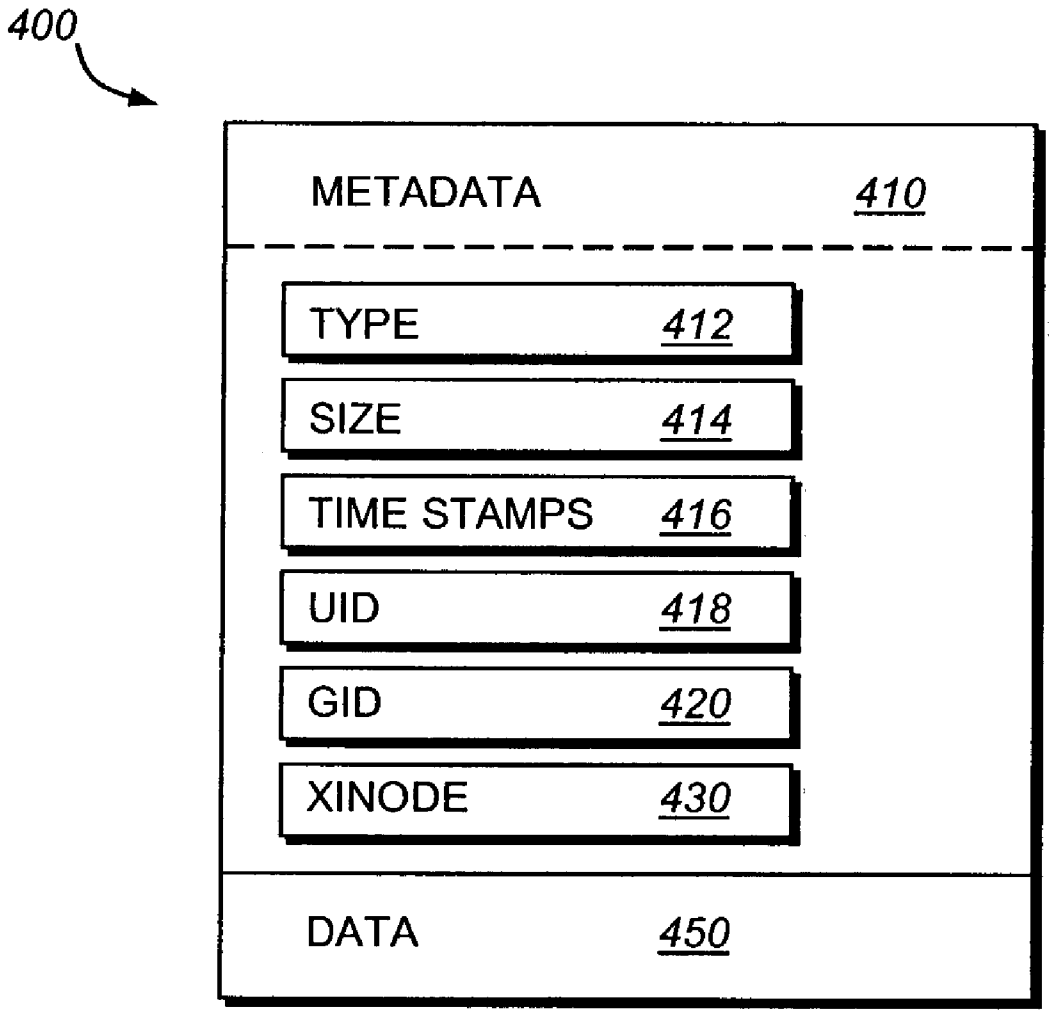

400
METADATA 410
TYPE 412
SIZE 414
TIME STAMPS 416
UID 418
GID 420
XINODE 430
DATA 450

FIG. 4

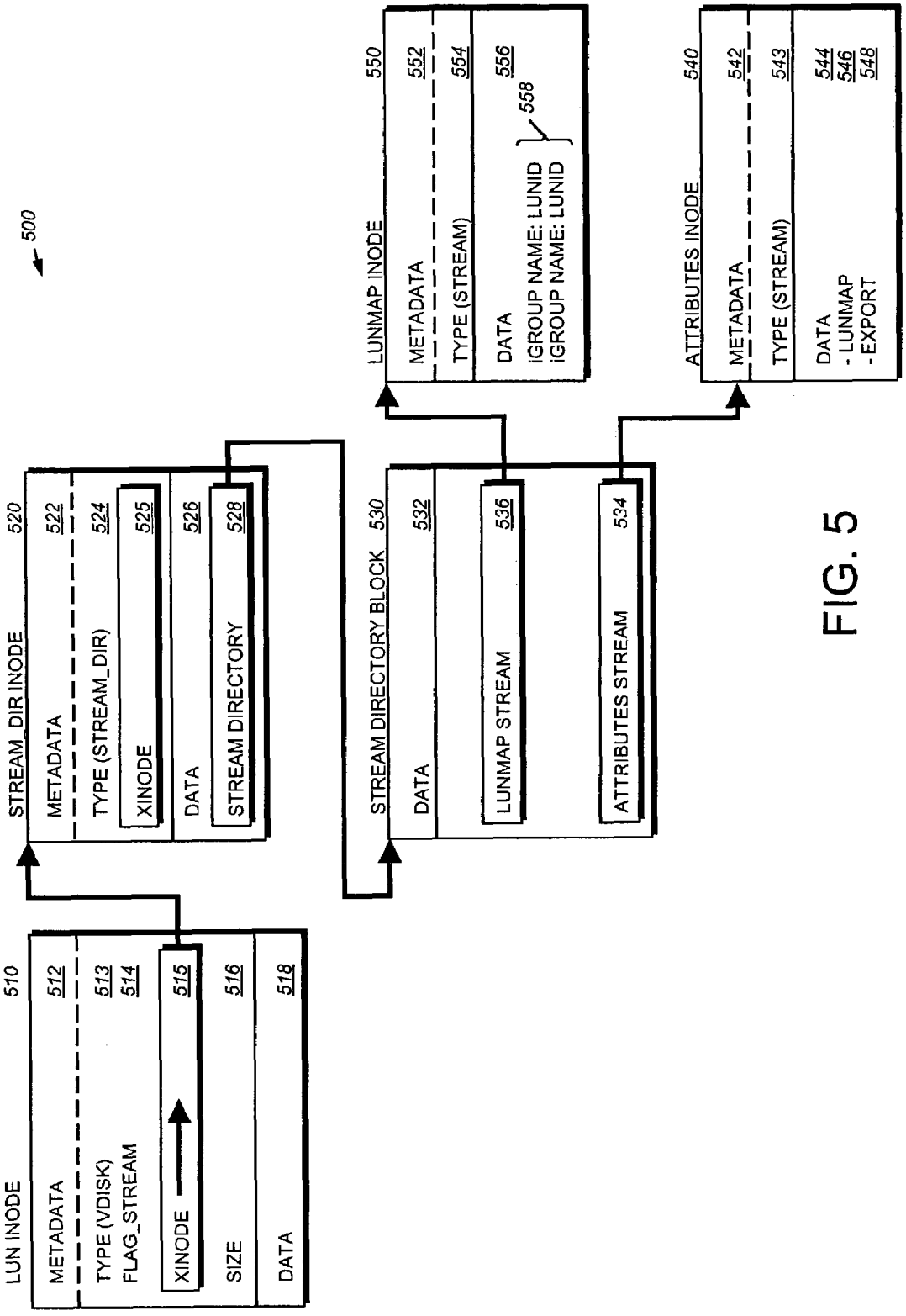

500
LUN INODE
510
METADATA
512
TYPE (VDISK)
FLAG_STREAM
513
514
XINODE
515
SIZE
516
DATA
518
STREAM_DIR INODE
520
METADATA
522
TYPE (STREAM_DIR)
524
XINODE
525
DATA
526
STREAM DIRECTORY
528
STREAM DIRECTORY BLOCK
530
DATA
532
LUNMAP STREAM
536
ATTRIBUTES STREAM
534
LUNMAP INODE
550
METADATA
552
TYPE (STREAM)
554
DATA
556
iGROUP NAME: LUNID
iGROUP NAME: LUNID
558
ATTRIBUTES INODE
540
METADATA
542
TYPE (STREAM)
543
DATA
- LUNMAP
- EXPORT
544
546
548

FIG. 5

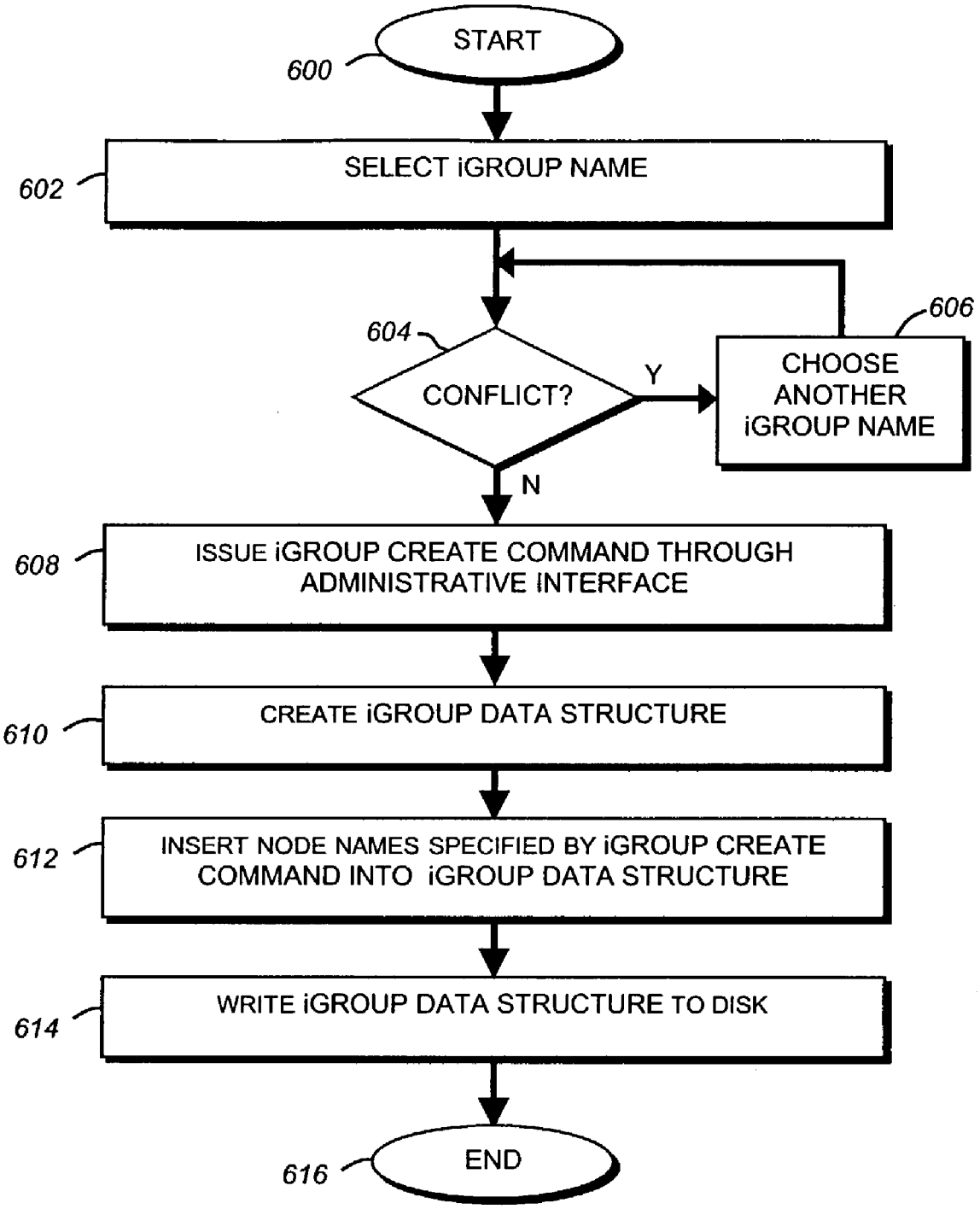

START
600
SELECT iGROUP NAME
602
604
CONFLICT?
Y
606
CHOOSE ANOTHER iGROUP NAME
N
608
ISSUE iGROUP CREATE COMMAND THROUGH ADMINISTRATIVE INTERFACE
610
CREATE iGROUP DATA STRUCTURE
612
INSERT NODE NAMES SPECIFIED BY iGROUP CREATE COMMAND INTO iGROUP DATA STRUCTURE
614
WRITE iGROUP DATA STRUCTURE TO DISK
616
END

FIG. 6

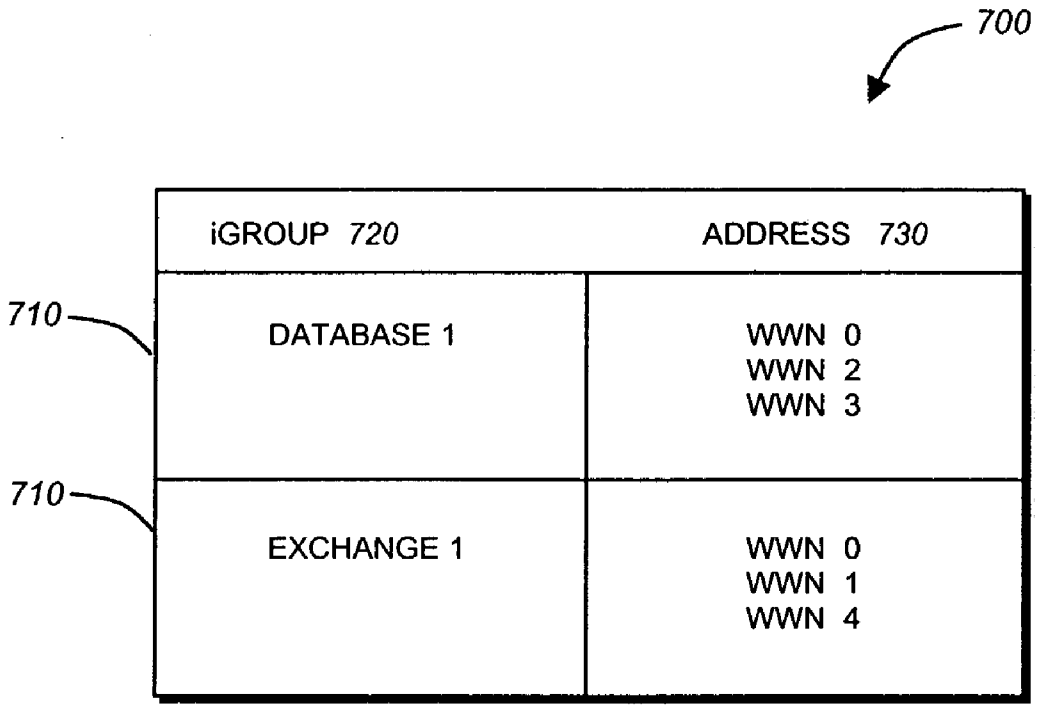

700
iGROUP 720
ADDRESS 730
710
DATABASE 1
WWN 0
WWN 2
WWN 3
710
EXCHANGE 1
WWN 0
WWN 1
WWN 4

FIG. 7

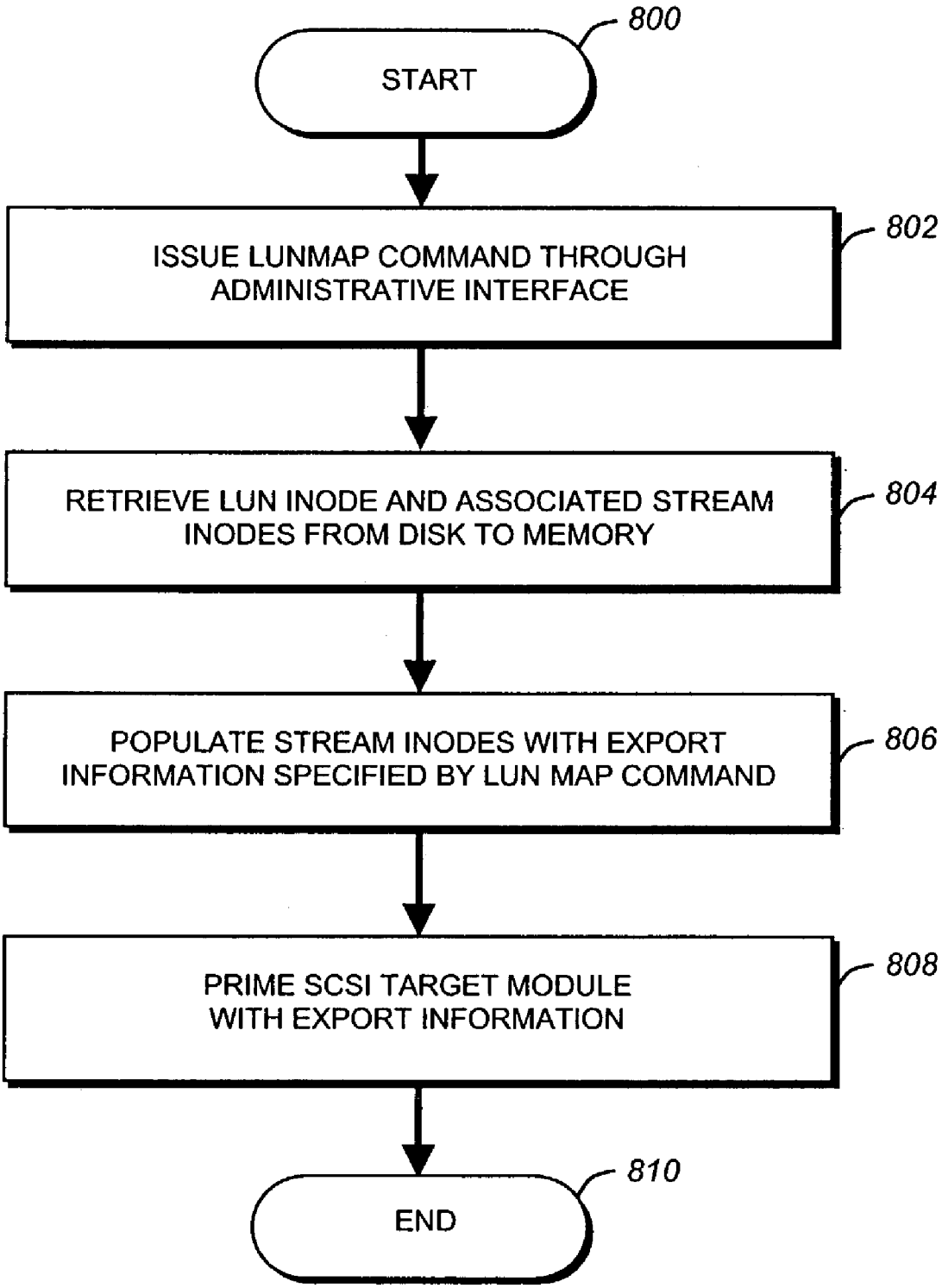

800
START
ISSUE LUNMAP COMMAND THROUGH ADMINISTRATIVE INTERFACE
802
RETRIEVE LUN INODE AND ASSOCIATED STREAM INODES FROM DISK TO MEMORY
804
POPULATE STREAM INODES WITH EXPORT INFORMATION SPECIFIED BY LUN MAP COMMAND
806
PRIME SCSI TARGET MODULE WITH EXPORT INFORMATION
808
END
810

FIG. 8

CONSISTENT LOGICAL NAMING OF INITIATOR GROUPS

FIELD OF THE INVENTION

The present invention relates to network storage systems and, more particularly, to accessing information stored on networked storage systems.

BACKGROUND OF THE INVENTION

A storage system is a computer that provides storage service relating to the organization of information on writable persistent storage devices, such as memories, tapes or disks. The storage system may be deployed within a storage area network (SAN) or a network attached storage (NAS) environment. When used within a NAS environment, the storage system may be embodied as a file server including an operating system that implements a file system to logically organize the information as a hierarchical structure of directories and files on, e.g., the disks. Each "on-disk" file may be implemented as a set of data structures, e.g., disk blocks, configured to store information, such as the actual data for the file. A directory, on the other hand, may be implemented as a specially formatted file in which information about other files and directories are stored.

The file server, or filer, may be further configured to operate according to a client/server model of information delivery to thereby allow many client systems (clients) to access shared resources, such as files, stored on the filer. Sharing of files is a hallmark of a NAS system, which is enabled because of semantic level of access to files and file systems. Storage of information on a NAS system is typically deployed over a computer network comprising a geographically distributed collection of interconnected communication links, such as Ethernet, that allow clients to remotely access the information (files) on the filer. The clients typically communicate with the filer by exchanging discrete frames or packets of data according to pre-defined protocols, such as the Transmission Control Protocol/Internet Protocol (TCP/IP).

In general, NAS systems utilize file-based protocols to access data stored on the filer. Each NAS client may therefore request the services of the filer by issuing file system protocol messages (in the form of packets) to the file system over the network. By supporting a plurality of file system protocols, such as the conventional Common Internet File System (CIFS), the Network File System (NFS) and the Direct Access File System (DAFS) protocols, the utility of the filer may be enhanced for networking clients.

A SAN is a high-speed network that enables establishment of direct connections between a storage system, such as an application server, and its storage devices. The SAN may thus be viewed as an extension to a storage bus and, as such, an operating system of the storage system enables access to stored information using block-based access protocols over the "extended bus". In this context, the extended bus is typically embodied as Fibre Channel (FC) or Ethernet media (i.e., network) adapted to operate with block access protocols, such as Small Computer Systems Interface (SCSI) protocol encapsulation over FC (FCP) or TCP/IP/Ethernet (iSCSI).

SCSI is a peripheral input/output (I/O) interface with a standard, device independent protocol that allows different peripheral storage devices, such as disks, to attach to the is storage system. These storage devices may be locally attached to the storage system or, in the case of a SAN

environment, attached via a network. In SCSI terminology, clients operating in a SAN environment are initiators that initiate requests and commands for data stored on the storage devices. The storage system is a target configured to respond to the requests issued by the initiators in accordance with a request/response protocol. The SAN clients typically identify and address the stored information in the form of blocks or disks by logical unit numbers ("luns").

In SCSI addressing, each initiator has a world wide name (WWN) or iSCSI name that is used by the initiator to access a lun entity, such as a disk, on the storage system. For example, each system on a FC SAN has a WWN, which is a 64-bit location independent identifier (ID) that is written in hexadecimal notation; iSCSI names are analogous to WWNs, but with a different format. The iSCSI names are used with the iSCSI protocol as IDs when restricting block-level access to a lun by one or more initiators. Thus, each time an initiator desires access to a lun, the WWN ID or iSCSI name must be provided to the storage system by a client/user of the initiator. This is an inefficient and possibly error-prone approach to accessing luns on a SAN storage system.

SUMMARY OF THE INVENTION

The present invention overcomes the disadvantages of the prior art by providing a technique that enables efficient access to logical unit numbers (luns) or virtual disks (vdisks) stored on a storage system, such as a multi-protocol storage appliance. The technique allows a grouping of initiators by a "human friendly" logical name that is mapped to a lun or vdisk on the storage appliance. By "human friendly" it is meant, generally, a hierarchical naming convention that may use a spoken language name including an arbitrary label selected by a user or administrator. The initiators are clients operating in, e.g., a storage area network (SAN) environment that initiate requests for the vdisk using block-based access protocols, such as the Small Computer Systems Interface (SCSI) protocol encapsulated over TCP/IP (iSCSI) or over fibre channel (FCP). The inventive technique enables access to the vdisk by all initiators that are members of the initiator group (igroup). An igroup is a logical named entity that is assigned to one or more addresses associated with one or more initiators. These addresses may comprise fibre channel (FC) world wide name (WWN) or iSCSI name identifiers (IDs). Therefore, rather than having to specify these IDs when desiring access to a vdisk, an initiator need only specify the human friendly name of the igroup.

According to the invention, the technique includes a method of creating logical igroups of initiators, each identified by a human-friendly name or label, and binding of each created igroup to one or more WWN or iSCSI IDs. An igroup may contain one initiator (in the case of a simple initiator-to-initiator group binding) or more initiators (in the case of a SAN cluster or a single client with multiple initiators for redundancy and/or multipathing purposes). In addition, the technique includes a method of assigning a lun ID to a vdisk and specifying the igroup of initiators that are allowed access to the vdisk, i.e., the clients to which the vdisk is exported. In other words, the igroup name is used to map a vdisk to all member initiators of the igroup. An initiator can be part of more than one igroup and inherit the vdisk (lun) mapping from all the groups.

An igroup has certain attributes, such as transport protocol type and operating system type of the member initiators. Illustratively, the igroup need not be homogeneous in terms of these attributes, i.e., an igroup can contain initiators

having different combinations of FCP and/or iSCSI as a transport. For example, iSCSI and FCP initiators can be combined into a single igroup. In addition, igroup can support various operating system initiator members. This allows operations, such as graceful rolling upgrade of a FCP SAN cluster to an iSCSI cluster, with no application downtime. Moreover, membership of the igroups can be modified at any time, i.e., initiators can be added to or removed from an igroup and, as a consequence, inherit or lose the mappings of the igroup, respectively.

Advantageously, the inventive technique obviates the need to use a WWN or iSCSI name ID during operation of the multi-protocol storage appliance except in a command used to bind the ID to an igroup. All other operations and commands invoked via, e.g., a user interface use only the igroup. This is a powerful and consistent abstraction with fundamental implications on global updates, e.g., when replacing an initiator in a client and the need to globally replace a WWN or iSCSI name, and in achieving goals of simplicity to create the multi-protocol storage appliance.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

FIG. **1** is a schematic block diagram of a multi-protocol storage appliance that may be advantageously used with the present invention;

FIG. **2** is a schematic block diagram of a storage operating system of the multiprotocol storage appliance that may be advantageously used with the present invention;

FIG. **3** is a schematic block diagram of a virtualization system that is implemented by a file system interacting with virtualization modules of the storage operating system;

FIG. **4** is a schematic block diagram of an on-disk inode data structure that may be advantageously used with the present invention;

FIG. **5** is a schematic block diagram illustrating an on-disk representation of virtual disk (vdisk) inode data structures, including logical unit number (lun) and lunmap inodes, in accordance with the present invention;

FIG. **6** is a flowchart illustrating a sequence of steps used to create an initiator group (igroup) in accordance with the present invention;

FIG. **7** is a schematic diagram of an igroup data structure having igroup definition entries in accordance with the present invention; and

FIG. **8** is a flowchart illustrating a sequence of steps used to map a lun to an igroup in accordance with the present invention.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

FIG. **1** is a schematic block diagram of a multi-protocol storage appliance **100** that may be advantageously used with the present invention. The multi-protocol storage appliance is configured to provide storage service for both file and block protocol access to information stored on storage devices in an integrated manner. In this context, the integrated multi-protocol appliance denotes a computer having features such as simplicity of storage service management and ease of storage reconfiguration, including reusable storage space, for users (system administrators) and clients of network attached storage is (NAS) and storage area network (SAN) deployments.

The multi-protocol storage appliance **100** is illustratively embodied as a storage system comprising a processor **122**, a memory **124**, a plurality of network adapters **125**, **126** and a storage adapter **128** interconnected by a system bus **123**. The multi-protocol storage appliance **100** also includes a storage operating system **200** that provides a virtualization system (and, in particular, a file system) to logically organize the information as a hierarchical structure of named directory, file and virtual disk (vdisk) storage objects on the disks **130**. An example of a multi-protocol storage appliance that may be advantageously used with the present invention is described in co-pending and commonly assigned U.S. patent application Ser. No. 10/215,917 titled A Multi-Protocol Storage Appliance that Provides Integrated Support for File and Block Access Protocols, which application is hereby incorporated by reference as though fully set forth herein.

Whereas clients of a NAS-based network environment have a storage viewpoint of files, the clients of a SAN-based network environment have a storage viewpoint of blocks or disks. To that end, the multi-protocol storage appliance **100** presents (exports) disks to SAN clients through the creation of logical unit numbers (luns) or vdisk objects. A vdisk object (hereinafter "vdisk") is a special file type that is implemented by the virtualization system and translated into an emulated disk as viewed by the SAN clients. The multi-protocol storage appliance thereafter makes these emulated disks accessible to the SAN clients through controlled exports.

In the illustrative embodiment, the memory **124** comprises storage locations that are addressable by the processor and adapters for storing software program code and data structures associated with the present invention. The processor and adapters may, in turn, comprise processing elements and/or logic circuitry configured to execute the software code and manipulate the data structures. The storage operating system **200**, portions of which are typically resident in memory and executed by the processing elements, functionally organizes the storage appliance by, inter alia, invoking storage operations in support of the storage service implemented by the appliance. It will be apparent to those skilled in the art that other processing and memory means, including various computer readable media, may be used for storing and executing program instructions pertaining to the invention described herein.

The network adapter **125** couples the storage appliance to a plurality of clients **160***a,b* over point-to-point links, wide area networks, virtual private networks implemented over a public network (Internet) or a shared local area network, hereinafter referred to as an illustrative Ethernet network **165**. For this NAS-based network environment, the clients are configured to access information stored on the multi-protocol appliance as files. Therefore, the network adapter **125** may comprise a network interface card (NIC) having the mechanical, electrical and signaling circuitry needed to connect the appliance to a network switch, such as a conventional Ethernet switch **170**. The clients **160** communicate with the storage appliance over network **165** by exchanging discrete frames or packets of data according to pre-defined protocols, such as the Transmission Control Protocol/Internet Protocol (TCP/IP).

The clients **160** may be general-purpose computers configured to execute applications over a variety of operating systems, including the Solaris™/ Unix® and Microsoft Windows® operating systems. Client systems generally

utilize file-based access protocols when accessing information (in the form of files and directories) over a NAS-based network. Therefore, each client **160** may request the services of the storage appliance **100** by issuing file access protocol messages (in the form of packets) to the appliance over the network **165**. For example, a client **160***a* running the Windows operating system may communicate with the storage appliance **100** using the Common Internet File System (CIFS) protocol over TCP/IP. On the other hand, a client **160***b* running the Solaris operating system may communicate with the multi-protocol appliance using either the Network File System (NFS) protocol over TCP/IP or the Direct Access File System (DAFS) protocol over a virtual interface (VI) transport in accordance with a remote DMA (RDMA) protocol over TCP/IP. It will be apparent to those skilled in the art that other clients running other types of operating systems may also communicate with the integrated multi-protocol storage appliance using other file access protocols.

The storage network "target" adapter **126** also couples the multi-protocol storage appliance **100** to clients **160** that may be further configured to access the stored information as blocks or disks. For this SAN-based network environment, the storage appliance is coupled to an illustrative Fibre Channel (FC) network **185**. FC is a networking standard describing a suite of protocols and media that is primarily found in SAN deployments. The network target adapter **126** may comprise a FC host bus adapter (HBA) having the mechanical, electrical and signaling circuitry needed to connect the appliance **100** to a SAN network switch, such as a conventional FC switch **180**. In addition to providing FC access, the FC HBA offloads fiber channel network processing operations for the storage appliance.

The clients **160** generally utilize block-based access protocols, such as the Small Computer Systems Interface (SCSI) protocol, when accessing information (in the form of blocks, disks or vdisks) over a SAN-based network. SCSI is a peripheral input/output (I/O) interface with a standard, device independent protocol that allows different peripheral devices, such as disks **130**, to attach to the storage appliance **100**. In SCSI terminology, clients **160** operating in a SAN environment are initiators that initiate requests and commands for data. The multi-protocol storage appliance is thus a target configured to respond to the requests issued by the initiators in accordance with a request/response protocol. The initiators and targets have endpoint addresses that, in accordance with the FC protocol, comprise worldwide names (WWN). A WWN is a unique identifier, e.g., a node name or a port name, consisting of an 8-byte number.

The multi-protocol storage appliance **100** supports various SCSI-based protocols used in SAN deployments, including SCSI encapsulated over TCP (iSCSI) and SCSI encapsulated over FC (FCP). The initiators (hereinafter clients **160**) may thus request the services of the target (hereinafter storage appliance **100**) by issuing iSCSI and FCP messages over the network **185** to access information stored on the disks. It will be apparent to those skilled in the art that the clients may also request the services of the integrated multi-protocol storage appliance using other block access protocols. By supporting a plurality of block access protocols, the multi-protocol storage appliance provides a unified and coherent access solution to vdisks/luns in a heterogeneous SAN environment.

The storage adapter **128** cooperates with the storage operating system **200** executing on the storage appliance to access information requested by the clients. The information may be stored on the disks **130** or other similar media adapted to store information. The storage adapter includes I/O interface circuitry that couples to the disks over an I/O interconnect arrangement, such as a conventional high-performance, FC serial link topology. The information is retrieved by the storage adapter and, if necessary, processed by the processor **122** (or the adapter **128** itself) prior to being forwarded over the system bus **123** to the network adapters **125**, **126**, where the information is formatted into packets or messages and returned to the clients.

Storage of information on the appliance **100** is preferably implemented as one or more storage volumes (e.g., VOL1-2 150) that comprise a cluster of physical storage disks **130**, defining an overall logical arrangement of disk space. The disks within a volume are typically organized as one or more groups of Redundant Array of Independent (or Inexpensive) Disks (RAID). RAID implementations enhance the reliability/integrity of data storage through the writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of redundant information with respect to the striped data. The redundant information enables recovery of data lost when a storage device fails.

Specifically, each volume **150** is constructed from an array of physical disks **130** that are organized as RAID groups **140**, **142**, and **144**. The physical disks of each RAID group include those disks configured to store striped data (D) and those configured to store parity (P) for the data, in accordance with an illustrative RAID 4 level configuration. It should be noted that other RAID level configurations (e.g. RAID 5) are also contemplated. In the illustrative embodiment, a minimum of one parity disk and one data disk may be employed. However, a typical implementation may include three data and one parity disk per RAID group and at least one RAID group per volume.

To facilitate access to the disks **130**, the storage operating system **200** implements a write-anywhere file system that cooperates with virtualization modules to provide a function that "virtualizes" the storage space provided by disks **130**. The file system logically organizes the information as a hierarchical structure of named directory and file objects (hereinafter "directories" and "files") on the disks. Each "on-disk" file may be implemented as set of disk blocks configured to store information, such as data, whereas the directory may be implemented as a specially formatted file in which names and links to other files and directories are stored. The virtualization system allows the file system to further logically organize information as a hierarchical structure of named vdisks on the disks, thereby providing an integrated NAS and SAN appliance approach to storage by enabling file-based (NAS) access to the files and directories, while further enabling block-based (SAN) access to the vdisks on a file-based storage platform.

In the illustrative embodiment, the storage operating system is preferably the NetApp® Data ONTAP™ operating system available from Network Appliance, Inc., Sunnyvale, Calif. that implements a Write Anywhere File Layout (WAFL™) file system. However, it is expressly contemplated that any appropriate storage operating system, including a write in-place file system, may be enhanced for use in accordance with the inventive principles described herein. As such, where the term "WAFL" is employed, it should be taken broadly to refer to any storage operating system that is otherwise adaptable to the teachings of this invention.

As used herein, the term "storage operating system" generally refers to the computer-executable code operable on a computer that manages data access and may, in the case of a multi-protocol storage appliance, implement data access

semantics, such as the Data ONTAP storage operating system, which is implemented as a microkernel. The storage operating system can also be implemented as an application program operating over a general-purpose operating system, such as Solaris or Windows, or as a general-purpose operating system with configurable functionality, which is configured for storage applications as described herein.

In addition, it will be understood to those skilled in the art that the inventive technique described herein may apply to any type of special-purpose (e.g., storage serving appliance) or general-purpose computer, including a standalone computer or portion thereof, embodied as or including a storage system. Moreover, the teachings of this invention can be adapted to a variety of storage system architectures including, but not limited to, a network-attached storage environment, a storage area network and disk assembly directly-attached to a client or host computer. The term "storage system" should therefore be taken broadly to include such arrangements in addition to any subsystems configured to perform a storage function and associated with other equipment or systems.

FIG. **2** is a schematic block diagram of the storage operating system **200** that may be advantageously used with the present invention. The storage operating system comprises a series of software layers organized to form an integrated network protocol stack or, more generally, a multi-protocol engine that provides data paths for clients to access information stored on the multi-protocol storage appliance using block and file access protocols. The protocol stack includes a media access layer **210** of network drivers (e.g., gigabit Ethernet drivers) that interfaces to network protocol layers, such as the IP layer **212** and its supporting transport mechanisms, the TCP layer **214** and the User Datagram Protocol (UDP) layer **216**. A file system protocol layer provides multi-protocol file access and, to that end, includes support for the DAFS protocol **218**, the NFS protocol **220**, the CIFS protocol **222** and the Hypertext Transfer Protocol (HTTP) protocol **224**. A VI layer **226** implements the VI architecture to provide direct access transport (DAT) capabilities, such as RDMA, as required by the DAFS protocol **218**.

An iSCSI driver layer **228** provides block protocol access over the TCP/IP network protocol layers, while a FC driver layer **230** operates with the FC HBA **126** to receive and transmit block access requests and responses to and from the integrated storage appliance. The FC and iSCSI drivers provide FC-specific and iSCSI-specific access control to the luns (vdisks) and, thus, manage exports of vdisks to either iSCSI or FCP or, alternatively, to both iSCSI and FCP when accessing a single vdisk on the multi-protocol storage appliance. In addition, the storage operating system includes a disk storage layer **240** that implements a disk storage protocol, such as a RAID protocol, and a disk driver layer **250** that implements a disk access protocol such as, e.g., a SCSI protocol.

Bridging the disk software layers with the integrated network protocol stack layers is a virtualization system **300**. FIG. **3** is a schematic block diagram of the virtualization system **300** that is implemented by a file system **320** interacting with virtualization modules illustratively embodied as, e.g., vdisk module **330** and SCSI target module **310**. It should be noted that the vdisk module **330**, the file system **320** and SCSI target module **310** can be implemented in software, hardware, firmware, or a combination thereof. The vdisk module **330** is layered on the file system **320** to enable access by administrative interfaces, such as a streamlined user interface (UI **350**), in response to a system administrator issuing commands to the multi-protocol storage appliance **100**. In essence, the vdisk module **330** manages SAN deployments by, among other things, implementing a comprehensive set of vdisk (lun) commands issued through the UI **350**, e.g., a command line interface (CLI **352**) or a graphical user interface (GUI **354**), by a system administrator. These vdisk commands are converted to primitive file system operations ("primitives") that interact with the file system **320** and the SCSI target module **310** to implement the vdisks.

The SCSI target module **310**, in turn, initiates emulation of a disk or lun by providing a mapping procedure that translates luns into the special vdisk file types. The SCSI target module is illustratively disposed between the FC and iSCSI drivers **228**, **230** and the file system **320** to thereby provide a translation layer of the virtualization system **300** between the SAN block (lun) space and the file system space, where luns are represented as vdisks **322**. To that end, the SCSI target module has a set of application programming interfaces (APIs **360**) that are based on the SCSI protocol and that enable a consistent interface to both the iSCSI and FCP drivers **228**, **230**. By "disposing" SAN virtualization over the file system **320**, the multi-protocol storage appliance reverses the approaches taken by prior systems to thereby provide a single unified storage platform for essentially all storage access protocols.

The file system **320** is illustratively a message-based system; as such, the SCSI target module **310** transposes a SCSI request into one or more messages representing an operation(s) directed to the file system. For example, a message generated by the SCSI target module may include a type of operation (e.g., read, write) along with a pathname (e.g., a path descriptor) and a filename (e.g., a special filename) of the vdisk object represented in the file system. Alternatively, the generated message may include an operation type and file handle containing volume/inode information. The SCSI target module **310** passes the message into the file system layer **320** as, e.g., a function call **365**, where the operation is performed.

The file system provides volume management capabilities for use in block-based access to the information, such as vdisks, stored on the storage devices, such as disks. That is, in addition to providing file system semantics, such as naming of storage objects, the file system **320** provides functions normally associated with a volume manager. These functions include (i) aggregation of the disks, (ii) aggregation of storage bandwidth of the disks, and (iii) reliability guarantees, such as mirroring and/or parity (RAID), to thereby present one or more storage objects layered on the file system. A feature of the multi-protocol storage appliance is the simplicity of use associated with these volume management capabilities, particularly when used in SAN deployments.

The file system **320** illustratively implements the WAFL file system having an on-disk format representation that is block-based using, e.g., 4 kilobyte (kB) blocks and using inodes to describe the files **324**. The WAFL file system uses files to store metadata describing the layout of its file system; these metadata files include, among others, an inode file. A file handle, i.e., an identifier that includes an inode number, is used to retrieve an inode from disk. A description of the structure of the file system, including ondisk inodes and the inode file, is provided in U.S. Pat. No. 5,819,292, titled Methodfor Maintaining Consistent States of a File System and for Creating User-Accessible Read-Only Copies of a

File System by David Hitz et al., issued Oct. 6, 1998, which patent is hereby incorporated by reference as though fully set forth herein.

FIG. **4** is a schematic block diagram illustrating an on-disk inode **400**, which includes a metadata section **410** and a data section **450**. The information stored in the metadata section **410** of each inode **400** describes the file and, as such, includes the type (e.g., regular or directory) **412** of file, the size 414 of the file, time stamps (e.g., access and/or modification) **416** for the file and ownership, i.e., user identifier (UID **418**) and group 1D (GID **420**), of the file. The metadata section **410** further includes a xinode field **430** containing a pointer that references another on-disk inode structure containing, e.g., access control list (ACL) information associated with the file or directory. The contents of the data section **450** of each inode, however, may be interpreted differently depending upon the type of file (inode) defined within the type field **412**. For example, the data section **450** of a directory inode contains metadata controlled by the file system, whereas the data section of a regular inode contains user-defined data. In this latter case, the data section **450** includes a representation of the data associated with the file.

Specifically, the data section **450** of a regular on-disk inode may include user data or pointers, the latter referencing 4 kB data blocks on disk used to store the user data. Each pointer is preferably a logical volume block number to thereby facilitate efficiency among the file system and the disk storage (RAID) layer **240** when accessing the data on disks. Given the restricted size (128 bytes) of the inode, user data having a size that is less than or equal to 64 bytes is represented, in its entirety, within the data section of that inode. However, if the user data is greater than 64 bytes but less than or equal to 64 kB, then the data section of the inode comprises up to 16 pointers, each of which references a 4 kB block of data on the disk. Moreover, if the size of the data is greater than 64 kilobytes but less than or equal to 64 megabytes (MB), then each pointer in the data section **450** of the inode references an indirect inode that contains 1024 pointers, each of which references a 4 kB data block on disk. Each data block is loaded from disk **130** into memory **124** in order to access the data. In addition, the size field **414** of the metadata section **410** of the inode refers to the size of the file.

Broadly stated, all inodes of the file system are organized into the inode file. A file system (FS) info block specifies the layout of information in the file system and includes an inode of a file that includes all other inodes of the file system. Each volume has an FS info block that is preferably stored at a fixed location within, e.g., a RAID group of the file system. The inode of the root FS info block may directly reference (point to) blocks of the inode file or may reference indirect blocks of the inode file that, in turn, reference direct blocks of the inode file. Within each direct block of the inode file are embedded inodes, each of which may reference indirect blocks that, in turn, reference data blocks of a file or vdisk.

Referring again to FIG. **3**, the file system implements access operations to vdisks **322**, as well as to files **324** and directories (dir **326**) that coexist with respect to global space management of units of storage, such as volumes **150** and/or qtrees **328**. A qtree **328** is a special directory that has the properties of a logical sub-volume within the namespace of a physical volume. Each file system storage object (file, directory or vdisk) is illustratively associated with one qtree, and quotas, security properties and other items can be assigned on a per-qtree basis. The vdisks and files/directories may be layered on top of qtrees **328** that, in turn, are layered on top of volumes **150** as abstracted by the file system "virtualization" layer **320**.

Note that the vdisk storage objects in the file system **320** are generally associated with SAN deployments of the multi-protocol storage appliance, whereas the file and directory storage objects are associated with NAS deployments of the appliance. The files and directories are generally not accessible via the FC or SCSI block access protocols; however, a file can be converted to a vdisk and then accessed by either the SAN or NAS protocol. The vdisks are thus accessible as luns from the SAN (FC and SCSI) protocols.

According to the invention, the UI **350** (CLI **352** and/or GUI **354**) interacts with the vdisk module **330** to introduce attributes and persistent lun map bindings that assign numbers to a created vdisk. These lun map bindings are thereafter used to export vdisks as certain SCSI identifiers (IDs) to the clients. In particular, the created vdisk can be exported via a lun mapping technique to enable a SAN client to "view" (access) a disk. Vdisks (luns) generally require strict controlled access in a SAN environment; sharing of luns in a SAN environment typically occurs only in limited circumstances, such as clustered file systems, clustered operating systems and multi-pathing configurations. A system administrator of the multi-protocol storage appliance determines which vdisks (luns) can be exported to a SAN client. Once a vdisk is exported as a lun, the client may access the vdisk over the SAN network utilizing a block access protocol, such as FCP and iSCSI.

SAN clients typically identify and address disks by logical numbers or luns. However, an "ease of management" feature of the multi-protocol storage appliance is that system administrators can manage vdisks and their addressing by logical names. To that end, the CLI **352** and/or GUI **354** interact with the vdisk module **330** and SCSI target module **310** to map logical names to vdisks. The present invention relates to a technique that allows a grouping of initiators by a "human friendly" logical name that is mapped to a lun or vdisk stored on the multi-protocol storage appliance to thereby enable access to the vdisk by all initiators that are members of the initiator group (igroup). As used herein, a "human friendly" logical name is an arbitrary label selected by the user of administrator that may be a spoken name, a path designation or include a hierarchical naming convention. An exemplary human friendly name would be "administrators" for a name of an igroup that comprises the administrators of a given network.

An igroup is a logical named entity that is assigned to one or more addresses, e.g., WWN or iSCSI name identifiers (IDs), associated with one or more initiators (depending upon whether a clustered environment is configured). The multi-protocol storage appliance manages export control of vdisks by logical names through the use of igroups. As described herein, an "igroup create" command essentially "binds" (associates) those addresses to a logical name or igroup. Therefore, rather than having to specify these IDs when desiring access to a vdisk, an initiator need only specify the human friendly name of the igroup.

As noted, a vdisk is a special file type in a volume that derives from a plain (regular) file, but that has associated export controls and operation restrictions that support emulation of a disk. Illustratively, the vdisk is a multi-inode object comprising a special file inode and a plurality of associated streaminodes, including an attributes stream inode and a lunmap streaminode. The special file (lun) inode functions as a data container for storing data, such as application data, associated with the emulated disk. The

lunmap inode contains a list of igroups to which the vdisk is exported, along with one or more addresses associated with one or more initiators that are assigned to each igroup.

FIG. **5** is a schematic block diagram illustrating an on-disk representation of vdisk inode data structures **500**, including a lun inode **510**, an attributes inode **540** and a lunmap inode **550**. As noted, the lun inode **510** is the special file inode that functions as a data container for storing application data associated with the vdisk **322**. That is, the lun inode comprises a data section **518** that may store the actual application data or pointers referencing 4 kB data blocks on disk used to store the data, as described in FIG. **4**. The data stored in this "default" data container can be retrieved (read) and stored (written) by a client using conventional block access protocols, such as the SCSI protocol. When appropriately configured, a vdisk may also be accessed using conventional file-level access protocols, such as the NFS protocol. In this configuration, a vdisk "appears" to be a regular file for such accesses. The lun inode **510** also comprises a metadata section **512** containing metadata such as the type **513** (i.e., a special vdisk type) and size 516 of the vdisk that, upon creation of the inode, is zero. A flag_stream flag **514** identifies the lun inode **510** as having one or more stream "sections", as provided by stream_dir inode **520**.

In order to access the stream_dir inode **520**, the pointer of xinode field **515** in lun inode **510** is modified to reference the inode **520**. The stream_dir inode **520** comprises a metadata section **522** that includes a type (stream_dir) field **524** and an xinode field **525** that references another on-disk inode structure containing, e.g., access control (such as CIFS permission) information associated with the vdisk. The inode **520** also includes a data section **526** containing a pointer **528** that references a stream directory data block associated with the vdisk, such as stream directory block **530**. The stream directory block **530** comprises a data section **532** that includes a plurality of entries, each containing an external representation of a streaminode along with mapping information (i.e., the inode number) for that inode. Two of those entries, entries **534** and **536**, contain mapping information (e.g., pointers) that reference an attributes (stream) inode **540** and a lunmap (stream) inode **550**, respectively.

The attributes inode **540** comprises a metadata section **542** that includes a type (stream) field **543** and a data section **544** that functions as a persistent store for holding various named attributes associated with the vdisk **322**. Attributes are an implementation mechanism that is internal to the file system and not managed by users. Examples of attributes include a lun map **546** and export information **548** controlling access to the vdisk by, e.g., specifying a list of initiators to which the vdisk is exported (i.e., those that have permissions to access to the vdisk). The lunmap inode **550** comprises a metadata section **552** that includes a type (stream) field **554** and a data section **556** that functions as a persistent store for holding a list **558** of name-value pairs. The name is illustratively an igroup name and the value is a lun identifier (ID). The vdisk and its associated inode data structures, including the attributes and lunmap inodes, are further described in co-pending and commonly assigned U.S. patent application Ser. No. 10/216,453 titled Storage Virtualization by Layering Vdisks on a File System, which application is hereby incorporated by reference as though fully set forth herein.

In the illustrative embodiment, vdisks (luns) are "layered" on top of igroups. The igroup abstracts the underlying details of "naming" (i.e., identification) of clients or initiators that desire access to the vdisks. The naming details (for purposes of allowing access to a vdisk/lun by a client/initiator) may be completely different between block access protocols, such as FCP and iSCSI. However, the logical naming of igroups is consistent with the FC and SCSI standards; the novel technique represents an application of those standards that simplifies access to the vdisks. The igroup abstraction thus decouples implementation of addressing from the underlying details of addressing. In other words, an igroup allows a user to define client or related clients addressing by logical names that are then used by higher layer vdisk (lun) commands to allow access. As a result, a vdisk (lun) can be easily shared over iSCSI and FCP, thereby allowing use in applications such as a mixed iSCSI or FCP cluster. Moreover, reorganization or upgrades of client/initiators do not affect security assignments (allowing access) at the lun level, since they are indirect via the logical igroup name.

According to the invention, the novel technique includes a method of creating logical igroups of initiators, each identified by a human-friendly name or label, and binding of each created igroup to one or more initiator addresses (e.g., WWN or iSCSI IDs). FIG. **6** is a flowchart illustrating a sequence of steps used to create an igroup in accordance with the present invention. The sequence starts at Step **600** and proceeds to Step **602** where a user (system administrator) selects a human-friendly name for the igroup. In Step **604**, a determination is made as to whether there is "clash" (conflict) between that selected name and an existing igroup name. If so, another igroup name is chosen in Step **606** and the sequence returns to Step **602** until there is no conflict.

In Step **608**, the user creates an igroup by issuing an "igroup create" command through, e.g., CLI **352**, GUI **354** or similar administrative interfaces associated with the multi-protocol storage appliance. An example of the igroup create command is:

igroup create-f<groupname> <nodename(s)>

wherein the <groupname> parameter indicates the human-friendly name of the igroup and the <nodename(s)> parameter indicates the initiator(s), e.g., the WWN or iSCSI address ID(s), bound to the igroup. An igroup may contain one initiator (in the case of a simple initiator-to-initiator group binding) or more initiators (in the case of a SAN cluster or a single client with multiple initiators for redundancy and/or multipathing purposes). The created igroup create thus enables grouping of initiators by a human friendly logical name.

In response to the igroup create command, the file system **320** cooperates with the vdisk module **330** to create an "in-core" igroup data structure **700** (e.g., a table in memory **124**) in Step **610**. In Step **612**, the vdisk module **330** processes the igroup create command to "call" primitive operations ("primitives") in the file system **320** to insert the name and addresses specified by the create command into the igroup data structure **700**. This step of the sequence essentially binds the logical, human-friendly name to one or more WWN or iSCSI IDs. Thereafter, in Step **614**, the igroup data structure is written (stored) to the disk **130**. The sequence then ends at Step **616**.

FIG. **7** is a schematic representation of the igroup data structure **700** having a plurality of igroup definition entries **710** that may be advantageously used with the present invention. Each definition **710** includes an igroup field **720** that identifies a particular human friendly igroup name and an address field **730** that identifies one or more addresses (e.g., WWN addresses or iSCSI IDs) associated with one or more initiators assigned to the igroup. For example, igroup "database **1**" is assigned a plurality of WWN 64-bit hexadecimal values 0, 2, 3, each of which associates a particular

initiator member to that igroup. Similarly, igroup "exchange **1**" is assigned WWN values 0, 1, 4, each associating an initiator member to that igroup.

The novel technique also includes a method of assigning a lun ID to a vdisk and specifying the igroup of initiators that are allowed access to the vdisk, i.e., the clients to which the vdisk is exported. In other words, this aspect of the inventive technique involves mapping a vdisk (lun) to an igroup. FIG. **8** is a flowchart illustrating a sequence of steps used to map a lun to an igroup in accordance with the present invention. The sequence starts at Step **800** and proceeds to Step **802** where a user maps a lun to an igroup by issuing a "lun map" command through the CLI **352**, GUI **354** or similar administrative interfaces associated with the multi-protocol storage appliance. An example of the lun map command is:

lun map <path> <groupname> <lunID>

wherein the <path> parameter is a path descriptor containing volume/inode information and a file handle to a vdisk, the <groupname> parameter indicates the human friendly name of the igroup and the <lunID> parameter indicates the lun ID assigned to the vdisk. Note that the lun ID is unique within the client's (initiator's) lun space. In Step **804**, the vdisk module **330** processes the lun map command to call primitives in the file system **320** to retrieve a lun inode and its associated streaminodes of the vdisk from disk, which are then loaded into memory (in-core). In Step **806**, primitives are called to populate the attributes inode **540** and lunmap inode **550** of the vdisk with the export information, including igroup and lun ID, specified by the lun map command. In Step **808**, the SCSI target module **310** is "primed" with the export information to enable access by the initiators of the igroup to the specified vdisk. The sequence then ends at Step **810**.

The igroup create and lun map commands essentially provide a mapping function between one or more initiators of an igroup and a lun ID that is representative of a vdisk. An initiator can be a member of more than one igroup and inherit the vdisk (lun) mapping from all the groups. The SCSI target module **310** thereafter implements the mapping function to allow a particular lun or vdisk to be exported to (accessed by) all initiators of a particular igroup. The mapping function is illustratively embodied as igroup definitions stored in the igroup data structure **700** and export information stored in the stream (lunmap and attributes) inodes. To that end, the SCSI target module is primed with export information, as indicated in Step **808**, by processing the contents of the igroup data structure and streaminodes.

Specifically, the lunmap streaminode **550** associated with each vdisk specifies the availability (visibility) of the vdisk to one or more igroups with respect to a lun ID. Since the igroups are identified by names associated with lun IDs within the lunmap inode, references are made to the igroup definitions **710** stored in the igroup data structure **700** in order to resolve each igroup name with a WWN address or iSCSI ID. The lun ID identified within the value portion of the name-value pair indicates the assigned lun number to the vdisk in accordance with a persistent lunmap "binding" that enables a SAN client to "view" (access) a vdisk.

According to the invention, the lun map command may be used to export one or more vdisks to the igroup, i.e., make the vdisk(s) "visible" to the igroup. In this sense, the lun map command is equivalent to an NFS export or a CIFS share. The WWN addresses or iSCSI IDs thus identify the igroups/clients that are allowed to access those vdisks specified by the lun map command. Thereafter, the logical igroup names are used with all operations internal to the storage operating system. This logical naming abstraction is pervasive throughout the entire vdisk command set, including interactions between a user and the multi-protocol storage appliance. In particular, the igroup naming convention is used for all subsequent export operations and listings of luns that are exported for various SAN clients.

Each igroup has certain associated attributes, such as transport protocol type and operating system type of its member initiators (node names). For example, the initiators within an igroup illustratively support the FCP, iSCSI and ATA transport protocol type. The operating system type attribute refers to the operating system type of the member initiators in the igroup. This latter attribute is provided because a SCSI target (such as the multiprotocol storage appliance) often "behaves" differently when interacting with different operating systems (e.g., Solaris, Windows, and Linux) of the initiators. Processing of a request received at the storage appliance occurs at lower layers of the storage operating system. Since addresses (WWN or iSCSI IDs) are bound to igroup name, these lower layers "know" whether the request is received over, e.g., an iSCSI or FCP transport.

In the illustrative embodiment of the present invention, the igroup need not be homogeneous in terms of these attributes, i.e., an igroup can contain initiators having different combinations of FCP, iSCSI and ATA as a transport. For example, iSCSI, FCP and ATA initiators may be combined into a single igroup. In addition, an igroup can support various operating system initiator members. This allows operations, such as graceful rolling upgrade of a FC SAN cluster to an iSCSI cluster, with no application downtime. In addition, membership of the igroups can be modified at any time, i.e., initiators can be added to or removed from an igroup and, as a consequence, inherit or lose the mappings of the igroup, respectively.

Use of the initiator group abstraction is particular advantageous in a situation where a particular initiator (client) may access more than one disk (vdisk) of the multiprotocol storage appliance for various applications. In that case, the user issues, in the following order, a lun create command through the UI that creates, e.g., twenty **100***g* vdisks called "vdisks**1-20**". The user then issues the igroup create command that binds a WWN of the client (initiator) to a logical name, e.g., "FCPClient". The WWN represents the address of a host bus adapter (HBA) module on the client and is thus assigned on a per module basis. The user then issues the lun map command that binds the vdisks**1-20** to the logical igroup name FCPClient.

Assume now that the HBA module in the client fails and a new adapter is installed in the client that has an address that is different from the address associated with the igroup FCPClient. Accordingly, the client can no longer access the vdisksl-20 because those disks are exported to a WWN address that is different from the new address of the client. To rectify this situation, a user need merely issue "igroup delete" and "igroup add" commands through the UI **350** of the multi-protocol storage appliance to reflect the new client HBA address. That change then "ripples" throughout the vdisk command set to reflect the new association of the igroup name to all of the vdisks**1-20**. This feature of the novel multi-protocol storage appliance architecture reflects global, single point changes to an igroup address for all the disks associated with that igroup.

In sum, the novel igroup technique decouples addressing from the actual protocol to thereby enable multi-protocol access to luns via either iSCSI or FCP, each of which has a different addressing scheme. Naming (e.g., hierarchical naming) simplifies management by hiding details of underlying addressing schemes of the transport. The igroup thus

comprises a multi-protocol encapsulation of host access control and address information making the vdisk transport protocol independent. That is, the vdisk can be simultaneously accessed by FCP, iSCSI or similar transports, such as ATA and serial ATA, with an abstract human friendly name hiding any addressing details. This feature provides high-level configuration information in the appliance the ability to handle any type of lower-level addressing. Therefore, igroup logical naming comprises (i) a pure naming and simplification arrangement, as well as (ii) an application thereof that cooperate to provide multi-protocol access control through a generalized initiator addressing technique.

Advantageously, the inventive technique obviates the need to use a WWN or iSCSI name ID during operation of the multi-protocol storage appliance except in a command used to bind the ID to an igroup. All other operations and commands invoked via, e.g., a user interface use only the igroup. This is a powerful and consistent abstraction with fundamental implications on global updates, e.g., when replacing an initiator in a client and the need to globally replace a WWN or iSCSI name, and in achieving goals of simplicity to create the multi-protocol storage appliance.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A method for enabling efficient access to a virtual disk (vdisk) stored on a storage system, the method comprising:

grouping initiators by a selectable name entered through an interface, where the initiators are clients that initiate a request for the vdisk using a block-based access protocol;

mapping the selectable name to the vdisk; and

enabling access to the vdisk using the selectable name by all initiators of the grouping.

2. The method of claim **1** wherein the step of grouping comprises:

issuing an initiator group (igroup) create command through the interface of the storage system, the igroup create command specifying the selectable name and one or more addresses of initiators bound to the name;

creating an igroup data structure in a memory of the storage system, where the igroup is a logical named entity that is assigned to the one or more addresses associated with the one or more initiators; and

inserting the name and initiator addresses specified by the igroup create command into the igroup data structure.

3. The method of claim **2** wherein the step of mapping comprises:

issuing a lun map command through the interface, the lunmap command specifying export information for the vdisk;

loading a lunmap inode of the vdisk into the memory; and

populating the lunmap inode with the export information specified by the lun map command.

4. The method of claim **3** wherein the export information comprises the name and a lun identifier (ID) assigned to the vdisk, and wherein the step of enabling comprises the step of priming a small computer systems interface module of the storage system with the name and lun ID.

5. The method of claim **4** wherein the step of priming comprises resolving the name associated with the lun ID to the initiator addresses stored in the igroup data structure.

6. The method of claim **1**, wherein the interface is an administrative interface.

7. The method of claim **1**, wherein the selectable name is an arbitrary label selected by a user.

8. A storage operating system configured to enable efficient access to a virtual disk (vdisk) stored on a storage device of a storage system, the storage operating system comprising:

a user interface adapted to receive commands that create an initiator group (igroup) of initiators and map the vdisk to the igroup, the commands specifying a user selected igroup name, addresses of the initiators and a logical unit number (lun) identifier (ID) assigned to the vdisk;

a file system configured to provide volume management capabilities for use in block-based access to the vdisk stored on the storage device;

a vdisk module that cooperates with the file system to bind the user selected igroup name to the initiator addresses; and

a small computer systems interface (SCSI) target module that cooperates with the vdisk module to map the user selected igroup name to the vdisk, the SCSI target module implementing a mapping function that allows the vdisk to be exported to the initiators of the igroup.

9. The storage operating system of claim **8** wherein storage system is a multiprotocol storage appliance and wherein the initiators are clients of the storage appliance operating in a storage area network (SAN) environment to initiate requests for the vdisk using block-based access protocols.

10. The storage operating system of claim **9** wherein the block-based access protocols comprise a SCSI protocol encapsulated over TCP/IP (iSCSI) or over fibre channel (FCP).

11. The storage operating system of claim **10** wherein an igroup is a logical named entity that is assigned to one or more addresses associated with one or more initiators.

12. The storage operating system of claim **11** wherein the addresses comprise FC world wide name (WWN) or iSCSI name IDs.

13. The storage operating system of claim **11** wherein the igroup comprises a multi-protocol encapsulation of host access control and address information making the vdisk specifically transport protocol independent.

14. The storage operating system of claim **8** wherein the file system further retrieves a lun inode and associated streaminodes of the vdisk from the storage device and loads the inodes into a memory of the storage system.

15. The storage operating system of claim **14** wherein one of the associated streaminodes of the vdisk embodies a lunmap inode that includes a data section functioning as a persistent store for holding a list of name-value pairs.

16. The storage operating system of claim **15** wherein each name-value pair stored in the lunmap inode comprises the igroup name and lun ID.

17. The storage operating system of claim **8** wherein the vdisk module cooperates with the file system to create an igroup data structure in a memory of the storage system, the vdisk module calling primitive operations in the file system to insert the user selected igroup name and initiator addresses into the igroup data structure.

18. Apparatus for enabling efficient access to a virtual disk (vdisk) stored on a storage system, the apparatus comprising:

means for grouping initiators by an initiator group (igroup) name, where the initiators are clients that initiate a request for the vdisk using a block-based access protocol;

means for mapping the igroup name to the vdisk; and

means for enabling access to the vdisk using the igroup name by all initiators of the grouping.

19. The apparatus of claim 18 wherein the means for grouping comprises:

means for issuing an igroup create command to the storage system, the igroup create command specifying the igroup name and one or more addresses of initiators bound to the igroup name;

means for creating an igroup data structure in a memory of the storage system; and

means for inserting the igroup name and initiator addresses specified by the igroup create command into the igroup data structure.

20. The apparatus of claim 19 wherein the means for mapping comprises:

means for issuing a lun map command to the storage system, the lunmap command specifying export information for the vdisk;

means for loading a lunmap inode of the vdisk into the memory; and

means for populating the lunmap inode with the export information specified by the lun map command.

21. A computer readable medium containing executable program instructions for enabling efficient access to a virtual disk (vdisk) stored on a storage system, the executable program instructions comprising program instructions for:

grouping initiators by an initiator group (igroup) name, where the initiators are clients that initiate a request for the vdisk using a block-based access protocol;

mapping the igroup name to the vdisk; and

enabling access to the vdisk using the igroup name by all initiators of the grouping.

22. The computer readable medium of claim 21 wherein the instruction for grouping comprises program instructions for:

issuing an igroup create command through an administrative interface of the storage system, the igroup create command specifying an igroup name and one or more addresses of initiators bound to the igroup name;

creating an igroup data structure in a memory of the storage system; and

inserting the igroup name and initiator addresses specified by the igroup create command into the igroup data structure.

23. The computer readable medium of claim 22 wherein the instruction for mapping comprises program instructions for:

issuing a lun map command through the administrative interface, the lunmap command specifying export information for the vdisk;

loading a lunmap inode of the vdisk into the memory; and

populating the lunmap inode with the export information specified by the lun map command.

24. The computer readable medium of claim 23 wherein the export information comprises the igroup name and lun identifier (ID) assigned to the vdisk, and wherein the instruction for enabling comprises program instructions for resolving the igroup name associated with the lun ID to the initiator addresses stored in the igroup data structure.

25. A method for faster access to a virtual disk (vdisk) stored on a storage system, the method comprising:

grouping one or more clients into an initiator group (igroup) entered through an interface, where the one or more clients initiate a request for the vdisk using a blockbased access protocol;

selecting a selectable name for the igroup, where the igroup is a logical named entity that is assigned to one or more addresses associated with the one or more clients; and

mapping the vdisk to the user selected name for the igroup to allow the one or more clients to access the vdisk using the selectable name.

26. The method of claim 25, further comprising:

issuing an igroup create command, the igroup create command specifying the user selected name and the one or more addresses of clients bound to the name;

creating an igroup data structure in a memory of the storage system; and

inserting the user selected name and client addresses specified by the igroup create command into the igroup data structure.

27. The method of claim 25, further comprising

issuing a lun map command, the lunmap command specifying export information for the vdisk;

loading a lunmap inode of the vdisk into a memory of the storage system; and

populating the lunmap inode with the export information specified by the lun map command.

28. The method of claim 25, further comprising:

implementing a mapping function that allows the vdisk to be exported to the one or more clients of the igroup by a small computer systems interface (SCSI) target module.

29. An apparatus for faster access to a virtual disk (vdisk) stored on a storage system, comprising:

means for grouping one or more clients into an initiator group (igroup), where the one or more clients initiate a request for the vdisk using a block-based access protocol;

means for selecting a selectable name for the igroup, where the igroup is a logical named entity that is assigned to one or more addresses associated with the one or more clients; and

means for mapping the vdisk to the user selected name for the igroup to allow the one or more clients to access the vdisk using the selectable name.

30. The apparatus of claim 29, further comprising:

means for issuing an igroup create command, the igroup create command specifying the user selected name and the one or more addresses of clients bound to the name;

means for creating an igroup data structure in a memory of the storage system; and

means for inserting the user selected name and client addresses specified by the igroup create command into the igroup data structure.

31. The apparatus of claim 29, further comprising

means for issuing a lun map command, the lunmap command specifying export information for the vdisk;

means for loading a lunmap inode of the vdisk into a memory of the storage sytstem; and

means for populating the lunmap inode with the export information specified by the lun map command.

32. The apparatus of claim 29, further comprising:

means for implementing a mapping function that allows the vdisk to be exported to the one or more clients of the igroup by a small computer systems interface (SCSI) target module.

33. A storage operating system configured to enable faster access to a virtual disk (vdisk) stored on a storage device, comprising:

a user interface adapted to receive commands that create an initiator group (igroup) of one or more clients, and to select a selectable name for the igroup, where the igroup is a logical named entity that is assigned to one or more addresses associated with the one or more clients; and

a vdisk module to bind the vdisk to the selectable name for the igroup to allow access to the one or more clients of the igroup using the user selected name.

34. The system of claim 33, further comprising:

a small computer systems interface (SCSI) target module that cooperates with the vdisk module to map the selectable name for the igroup to the vdisk, the SCSI target module implements a mapping function that allows the vdisk to be exported to the one or more clients of the igroup.

35. An improved method for access to a virtual disk (vdisk) stored on a storage system, the method comprising:

receiving a block-based access protocol first request from one or more clients with the first request including one or more addresses to the vdisk;

grouping, in response to the first request, the one or more clients by a user selected name to form a group;

binding the one or more addresses to the group with the user selected name;

mapping the group with the user selected name to the vdisk; and

receiving, from a client of the group, a second request using the user selected name to access the vdisk.

36. The method of claim 35, wherein the one or more addresses is at least one of a fibre channel world wide name and an iSCSI name identifier.

37. The method of claim 35, wherein the step of grouping comprises the steps of:

issuing a group create command through an administrative interface of the storage system, the group create command specifying the user selected name and one or more addresses of clients bound to the name;

creating a group data structure in a memory of the storage system; and

inserting the name and the one or more addresses specified by the group create command into the group data structure.

38. The method of claim 35, wherein the step of mapping comprises the steps of:

issuing a lunmap command through an administrative interface, the lunmap command specifying export information for the vdisk;

loading a lunmap inode of the vdisk into the memory; and

populating the lunmap inode with the export information specified by the lunmap command.

* * * * *

Exhibit D

US007328305B2

(12) **United States Patent**
Kleiman et al.

(10) **Patent No.: US 7,328,305 B2**
(45) **Date of Patent: Feb. 5, 2008**

(54) **DYNAMIC PARITY DISTRIBUTION TECHNIQUE**

(75) Inventors: **Steven R. Kleiman**, Los Altos, CA (US); **Robert M. English**, Menlo Park, CA (US); **Peter F. Corbett**, Lexington, MA (US)

(73) Assignee: **Network Appliance, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 409 days.

(21) Appl. No.: **10/700,227**

(22) Filed: **Nov. 3, 2003**

(65) **Prior Publication Data**

US 2005/0097270 A1 May 5, 2005

(51) **Int. Cl.**
G06F 12/00 (2006.01)

(52) **U.S. Cl.** **711/114**; 714/6; 714/758; 714/800; 714/801; 714/802; 714/803; 714/804; 714/805

(58) **Field of Classification Search** 711/113, 711/114; 714/5, 6, 701, 770, 800
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,876,978	A	4/1975	Bossen et al.
4,092,732	A	5/1978	Ouchi
4,201,976	A	5/1980	Patel
4,205,324	A	5/1980	Patel
4,375,100	A	2/1983	Tsuji et al.
4,467,421	A	8/1984	White
4,517,663	A	5/1985	Imazeki et al.
4,667,326	A	5/1987	Young et al.
4,688,221	A	8/1987	Nakamura et al.
4,722,085	A	1/1988	Flora et al.
4,755,978	A	7/1988	Takizawa et al.
4,761,785	A	8/1988	Clark et al.
4,775,978	A	10/1988	Hartness
4,796,260	A	1/1989	Schilling et al.
4,817,035	A	3/1989	Timsit
4,825,403	A	4/1989	Gershenson et al.
4,837,680	A	6/1989	Crockett et al.

(Continued)

OTHER PUBLICATIONS

Patterson et al., A Case for Redundant Arrays of Inexpensive Disks (RAID), □□Dept. of Elect. Engr. and Computer Sciences, Univ. of Cal., Berkeley, 1988 ACM □□0-89791-268-3/88/006/0109, pp. 109-116.*

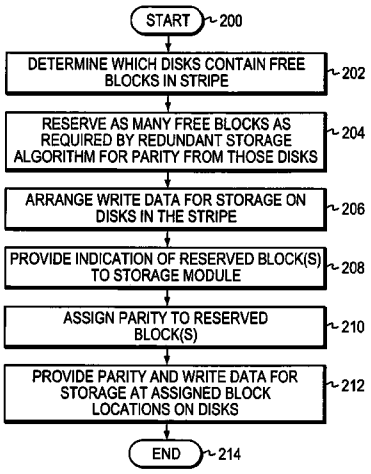
(Continued)

Primary Examiner—Matthew Kim
Assistant Examiner—Hetul Patel
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP

(57) **ABSTRACT**

A dynamic parity distribution system and technique distributes parity across disks of an array. The dynamic parity distribution system includes a storage operating system that integrates a file system with a RAID system. In response to a request to store (write) data on the array, the file system determines which disks contain free blocks in a next allocated stripe of the array. There may be multiple blocks within the stripe that do not contain file system data (i.e., unallocated data blocks) and that could potentially store parity. One or more of those unallocated data blocks can be assigned to store parity, arbitrarily. According to the dynamic parity distribution technique, the file system determines which blocks hold parity each time there is a write request to the stripe. The technique alternately allows the RAID system to assign a block to contain parity when each stripe is written.

51 Claims, 2 Drawing Sheets



U.S. PATENT DOCUMENTS

4,847,842	A	7/1989	Schilling	
4,849,929	A	7/1989	Timsit	
4,849,974	A	7/1989	Schilling et al.	
4,849,976	A	7/1989	Schilling et al.	
4,870,643	A	9/1989	Bultman et al.	
4,899,342	A	2/1990	Potter et al.	
4,989,205	A	1/1991	Dunphy, Jr. et al.	
4,989,206	A	1/1991	Dunphy, Jr. et al.	
5,077,736	A	12/1991	Dunphy, Jr. et al.	
5,088,081	A	2/1992	Farr	
5,101,492	A	3/1992	Schultz et al.	
5,128,810	A	7/1992	Halford	
5,148,432	A	9/1992	Gordon et al.	
RE34,100	E	10/1992	Hartness	
5,163,131	A	11/1992	Row et al.	
5,166,936	A	11/1992	Ewert et al.	
5,179,704	A	1/1993	Jibbe et al.	
5,202,979	A	4/1993	Hillis et al.	
5,208,813	A	5/1993	Stallmo	
5,210,860	A	5/1993	Pfeffer et al.	
5,218,689	A	6/1993	Hotle	
5,233,618	A	8/1993	Glider et al.	
5,235,601	A	8/1993	Stallmo et al.	
5,237,658	A	8/1993	Walker et al.	
5,257,367	A	10/1993	Goodlander et al.	
5,274,799	A	12/1993	Brant et al.	
5,305,326	A	4/1994	Solomon et al.	
5,351,246	A	9/1994	Blaum et al.	
5,410,667	A	4/1995	Belsan et al.	
5,537,567	A	7/1996	Galbraith et al.	
5,579,475	A	11/1996	Blaum et al.	
5,623,595	A	4/1997	Bailey	
5,657,468	A *	8/1997	Stallmo et al.	711/114
5,805,788	A	9/1998	Johnson	
5,812,753	A	9/1998	Chiariotti	
5,862,158	A	1/1999	Baylor et al.	
5,884,098	A	3/1999	Mason, Jr.	
6,092,215	A	7/2000	Hodges et al.	
6,138,201	A	10/2000	Rebalski	
6,158,017	A	12/2000	Han et al.	
6,223,300	B1	4/2001	Gotoh	
6,532,548	B1	3/2003	Hughes	
6,557,123	B1 *	4/2003	Wiencko et al.	714/701
6,571,326	B2 *	5/2003	Spiegel et al.	711/170
6,581,185	B1	6/2003	Hughes	
2002/0083037	A1	6/2002	Lewis et al.	
2002/0124137	A1 *	9/2002	Ulrich et al.	711/113
2003/0126523	A1	7/2003	Corbett et al.	

OTHER PUBLICATIONS

William R. Stanek. "Microsoft Windows 2000 Server: Administering Volume Sets and RAID Arrays" Jan. 10, 2006. (Printed 1999)□□(http://www.microsoft.com/technet/prodtechnol/windows2000serv/maintain/operate/11w2kada.mspx).*

Microsoft Computer Dictionary, 5th Edition, 2002, p. 211.*

Anvin, Peter H, "*The Mathematics of RAID 6*," Dec. 2004.

Auspex 4Front NS2000, System Architecture, Network-Attached Storage For a New Millennium, Auspex Engineering Technical Report 24, Jan. 1999.

Bestavros, Azer, et al., *Reliability and Performance of Parallel Disks*, Technical Memorandum 45312-891206-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, Dec. 1989.

Bitton, Dina, *Disk Shadowing*, Proceedings of the 14th VLDB Conference, LA, CA (1988).

Bultman, David L., *High Performance SCSI Using Parallel Drive Technology*, In Proc. BUSCON Conf., pp. 40-44, Anaheim, CA, Feb. 1988.

Chen, Peter et al., *Two Papers on RAIDs*. Technical Report, CSD-88-479, Computer Science Division, Electrical Engineering and Computer Sciences, University of California at Berkeley (1988).

Chen, Peter M., et al., *An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890*, Performance Evaluation, pp. 74-85, 1990—check to see if exact same copy as one in WAFL.

Chen, Peter M.., et al, Maximizing Performance in a Striped Disk Array, Proc. 1990 *ACM SIGARCH 17th Intern*. Symp. on Comp. Arch., Seattle, WA, May 1990, pp. 322-331.

Chen, Peter M., et al., *RAID:High Performance, Reliable Secondary Storage*, ACM Computing Surveys, 26(2):145-185, Jun. 1994.

Chervenak, Ann L., *Performance Measurement of the First RAID Prototype*, Technical Report UCB/CSD 90/574, Computer Science Division (EECS), University of California, Berkeley, May 1990.

Copeland, George, et al., "*A Comparison of High-Availability Media Recovery techniques*," in Proc. ACM-SIGMOD Int. Conf. Management of Data, 1989.

Courtright II, William V., et al., *RAIDframe: A Rapid Prototyping Tool for RAID Systems*, Computer Science Technical Report CMU-CS97-142, Carnegie Mellon University, Pittsburgh, PA 15213, Jun. 4, 1997.

Evans *The Tip of the Iceberg:RAMAC Virtual Array—Part I*, Technical Support, Mar. 1997, pp. 1-4.

Kim, Michelle Y., *Synchronized Disk Interleaving*, IEEE Transactions on Computers, C-35(11):978-988, Nov. 1986.

Kim, Michelle, et al., *Asynchronous Disk Interleaving Approximating Access Delays*, IEEE Transactions on Computers, vol. 40, No. 7, Jul. 1991, pp. 801-810.

Lawlor, F. D., *Efficient Mass Storage Parity Recovery Mechanism*, IBM Technical Disclosure Bulletin 24(2):986-987, Jul. 1981.

Lee, Edward K., et al., *RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service*, Technical Report UCB/CSD 92/672, (Feb. 1992).

Li, Don, et al., *Authors' Reply*, IEEE Transactions on Communications, 46:575, May 1998.

Livny, Miron, et al., *Multi-Disk Management Algorithms*, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pp. 69-77, Banff, Alberta, Canada, May 1987.

Meador, Wes E., *Disk Array Systems*, Proceedings of COMPCON, 1989, pp. 143-146.

Ng, Spencer, et al., *Trade-Offs Between Devices and Paths in Achieving Disk Interleaving*, IEEE International Symposium on Computer Architecture, 1988, pp. 196-201.

Ng, Spencer, *Some Design Issues of Disk Arrays*, Proceedings of COMPCON Spring '89, pp. 134-142. IEEE, 1989.

Park, Arvin, et al., *Providing Fault Tolerance In Parallel Secondary Storage Systems*, Technical Reports CS-TR-057-86, Princeton, Nov. 1986.

Patel, Arvind M., *Adaptive Cross-Parity (AXP) Code for a High-Density Magnetic Tape Subsystem*, IBM Technical Disclosure Bulletin 29(6):546-562, Nov. 1985.

Patterson, D., et al., *A Case for Redundant Arrays of Inexpensive Disks (RAID)*,_Technical Report, CSD-87-391, Computer Science Division, Electrical Engineering and Computer Sciences, University of California at Berkeley (1987).

Patterson, D., et al., *A Case for Redundant Arrays of Inexpensive Disks (RAID)*,_SIGMOD International Conference on Management of Data, Chicago, IL, USA, Jun. 1-3, 1988, SIGMOD RECORD (17)3:109-16 (Sep. 1988).

Gibson, Garth A., et al., *Coding Techniques for Handling Failures in Large Disk Arrays*, Technical Report UCB/CSD 88/477, Computer Science Division, University of California, (Jul. 1988.).

Gibson, Garth A., et al., *Failure Correction Techniques for Large Disk Arrays*, In Proceedings Architectural Support for Programming Languages and Operating Systems, Boston, Apr. 1989, pp. 123-132.

Gibson, Garth A., et al., *Strategic Directions in Storage I/O Issues in Large-Scale Computing*, ACM Computing Survey, 28(4):779-93, Dec. 1996.

Goldick, Jonathan S., et al., *Multi-resident AFS: An Adventure in Mass Storage*, In Proceedings of the 1995 USENIX Technical Conference, pp. 47-58, Jan. 1995.

Graham, Susan L., et al., *Massive Information Storage, Management, and Use*, (NSF Institutional Infrastructure Proposal), Technical Report No. UCB/CSD 89/493, Jan. 1989.

Gray, Jim et al., *Parity striping of disc arrays: Low-Cost Reliable Storage with Acceptable Throughput*. In Proceedings of the 16th Very Large Data Bases Conference, pp. 148-161, Brisbane, Australia, 1990.

Grimes, DW Martinez, *Two Dimensional Parity Error Correction Procedure*, IBM Technical Disclosure Bulletin 2686-2689, Oct. 1982.

Grimes, DW Martinez, *Vertical Parity Generator for Two Dimensional Parity*, IBM Technical Disclosure Bulletin 2682-2685, Oct. 1982.

Hellerstein, Lisa, et al,. *Coding Techniques for Handling Failures in Large Disk Arrays*. In Algorithmica vol. 2, Nr. 3, 182-208 (1994).

Hughes, James, et al., High Performance RAIT, *Tenth NASA Goddard Conference on Mass Storage Systems and Technologies and Nineteenth IEEE Symposium on Mass Storage Systems*, Adelphi, Maryland, USA, Apr. 2002.

Johnson, Theodore, et al, *Tape Group Parity Protection*, IEEE Symposium on Mass Storage, pp. 72-79, Mar. 1999.

Katz, Randy H. et al., *Disk System Architectures for High Performance Computing*, Dec. 1989.

Kent, Jack et al., Optimizing Shadow Recovery Algorithms, *IEEE Transactions on Software Engineering*, 14(2):155-168, Feb. 1988.

Patterson, David A., et al., *Introduction to Redundant Arrays of Inexpensive Disks (RAID)*. In IEEE Spring 89 COMPCON, San Francisco, IEEE Computer Society Press, Feb. 27-Mar. 3, 1989, pp. 112-117.

Storagesuite "Performance Without Compromise: The Virtual Storage Architecture,"catalogue, 1997.

Reddy, A. L. Narasimha, et al., *An Evaluation of Multiple-Disk I/O Systems*, IEEE Transactions on Computers, vol. 38, No. 12, Dec. 1989, pp. 1680-1690.

Schulze, Martin E., *Considerations in the Design of a RAID Prototype*, Computer Science Division, Department of Electrical Engineering and Computer Sciences, Univ. of CA, Berkley, Aug. 25, 1988.

Schulze, Martin., et al., *How Reliable is a RAID?* , Proceedings of COMPCON, 1989, pp. 118-123.

Shirriff, Kenneth W., *Sawmill:A Logging File System for a High-Performance RAID Disk Array*, CSD-95-862, Jan. 1995.

Stonebraker, Michael, et al., *The Design of XPRS*, Proceedings of the 14th VLDB Conference, LA, CA (1988).

Tanabe, Takaya, et al, *Redundant Optical Storage System Using DVD-RAM Library*, IEEE Symposium on Mass Storage, pp. 80-87, Mar. 1999.

Tekrom—"About RAID 6", 2004.

Tweten, David, *Hiding Mass Storage Under UNIX: NASA's MSS-H Architecture*, IEEE Symposium on Mass Storage, pp. 140-145, May 1990.

Wilkes, John, et al., *The HP AutoRAID hierarchical storage system*, ACM Transactions on Computer Systems, Feb. 1996, vol. 14, pp. 108-136.

* cited by examiner

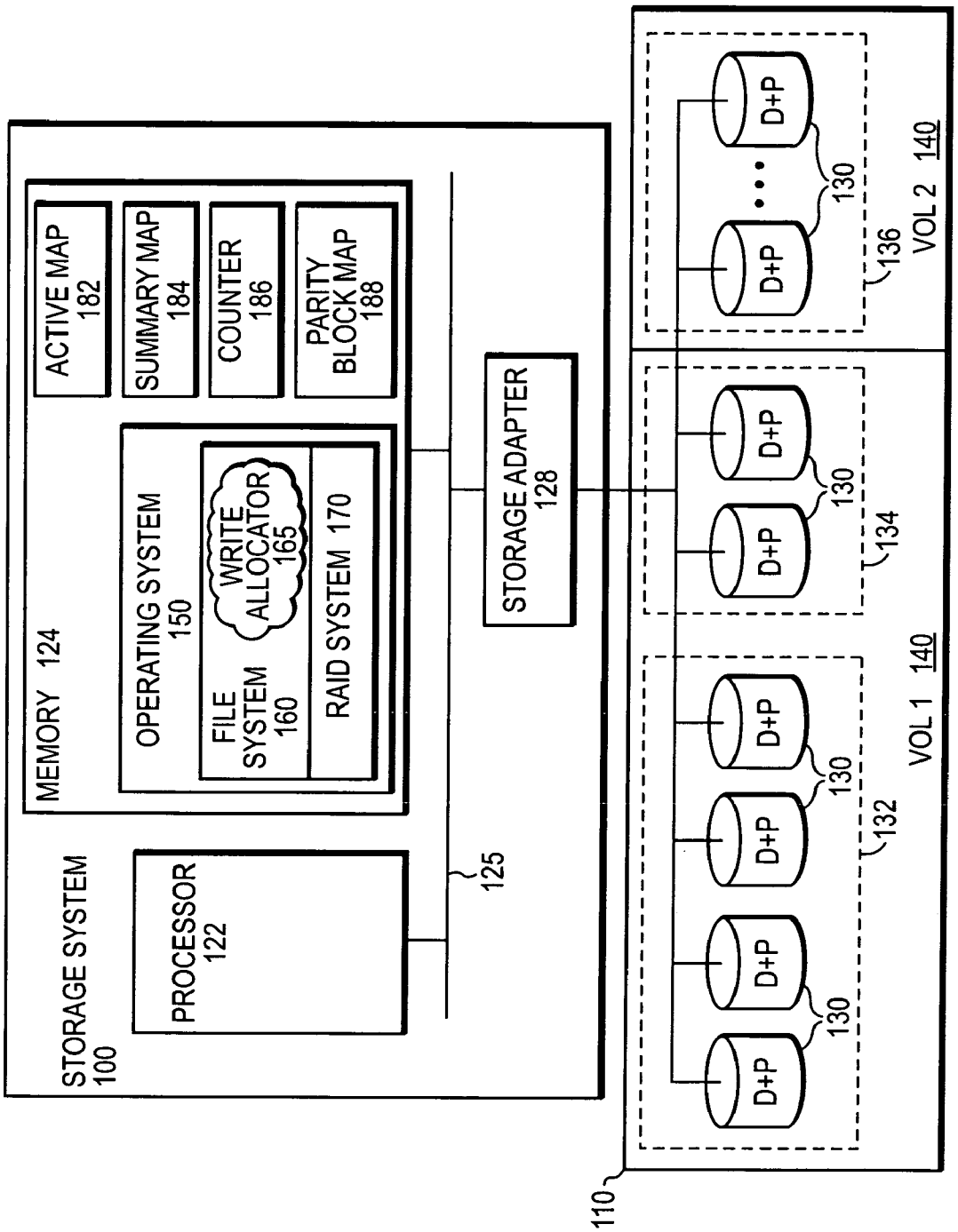

STORAGE SYSTEM
100
PROCESSOR
122
MEMORY 124
ACTIVE MAP
182
SUMMARY MAP
184
COUNTER
186
PARITY
BLOCK MAP
188
OPERATING SYSTEM
150
FILE
SYSTEM
160
WRITE
ALLOCATOR
165
RAID SYSTEM 170
125
STORAGE ADAPTER
128
110
VOL 1 140
VOL 2 140
D+P
130
132
134
136

FIG. 1

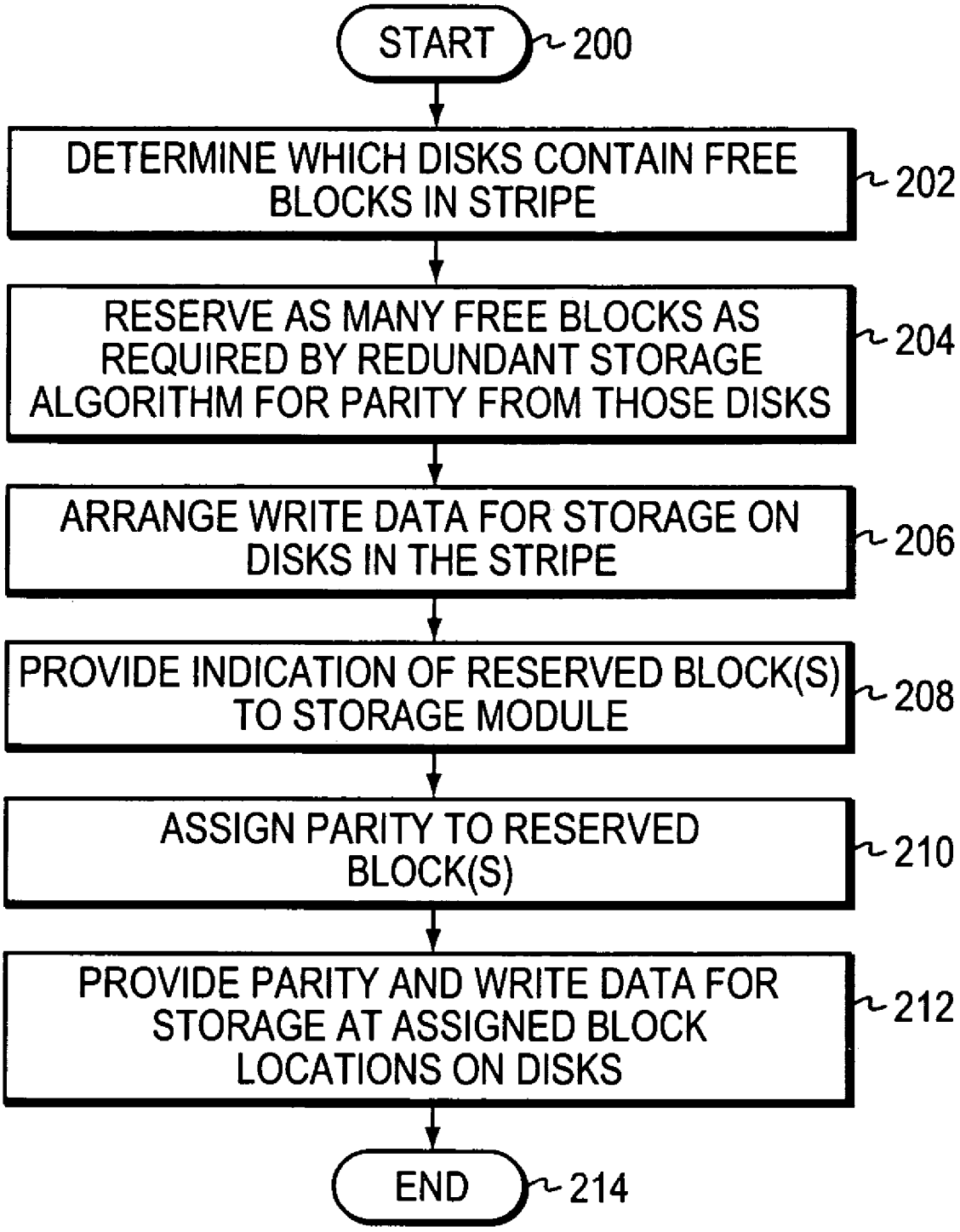

START
200
DETERMINE WHICH DISKS CONTAIN FREE BLOCKS IN STRIPE
202
RESERVE AS MANY FREE BLOCKS AS REQUIRED BY REDUNDANT STORAGE ALGORITHM FOR PARITY FROM THOSE DISKS
204
ARRANGE WRITE DATA FOR STORAGE ON DISKS IN THE STRIPE
206
PROVIDE INDICATION OF RESERVED BLOCK(S) TO STORAGE MODULE
208
ASSIGN PARITY TO RESERVED BLOCK(S)
210
PROVIDE PARITY AND WRITE DATA FOR STORAGE AT ASSIGNED BLOCK LOCATIONS ON DISKS
212
END
214

FIG. 2

DYNAMIC PARITY DISTRIBUTION TECHNIQUE

FIELD OF THE INVENTION

The present invention relates to arrays of storage systems and, more specifically, to a system that efficiently assigns parity blocks within storage devices of a storage array.

BACKGROUND OF THE INVENTION

A storage system typically comprises one or more storage devices into which information may be entered, and from which information may be obtained, as desired. The storage system includes a storage operating system that functionally organizes the system by, inter alia, invoking storage operations in support of a storage service implemented by the system. The storage system may be implemented in accordance with a variety of storage architectures including, but not limited to, a network-attached storage environment, a storage area network and a disk assembly directly attached to a client or host computer. The storage devices are typically disk drives organized as a disk array, wherein the term "disk" commonly describes a self-contained rotating magnetic media storage device. The term disk in this context is synonymous with hard disk drive (HDD) or direct access storage device (DASD).

Storage of information on the disk array is preferably implemented as one or more storage "volumes" that comprises a cluster of physical disks, defining an overall logical arrangement of disk space. The disks within a volume are typically organized as one or more groups, wherein each group may be operated as a Redundant Array of Independent (or Inexpensive) Disks (RAID). In this context, a RAID group is defined as a number of disks and an address/block space associated with those disks. The term "RAID" and its various implementations are well-known and disclosed in *A Case for Redundant Arrays of Inexpensive Disks* (*RAID*), by D. A. Patterson, G. A. Gibson and R. H. Katz, Proceedings of the International Conference on Management of Data (SIGMOD), June 1988.

The storage operating system of the storage system may implement a high-level module, such as a file system, to logically organize the information as a hierarchical structure of directories, files and blocks on the disks. For example, each "on-disk" file may be implemented as set of data structures, i.e., disk blocks, configured to store information, such as the actual data for the file. The storage operating system may also implement a storage module, such as a disk array controller or RAID system, that manages the storage and retrieval of the information to and from the disks in accordance with write and read operations. There is typically a one-to-one mapping between the information stored on the disks in, e.g., a disk block number space, and the information organized by the file system in, e.g., volume block number space.

A common type of file system is a "write in-place" file system, an example of which is the conventional Berkeley fast file system. In a write in-place file system, the locations of the data structures, such as data blocks, on disk are typically fixed. Changes to the data blocks are made "in-place"; if an update to a file extends the quantity of data for the file, an additional data block is allocated. Another type of file system is a write-anywhere file system that does not overwrite data on disks. If a data block on disk is retrieved (read) from disk into a memory of the storage system and "dirtied" with new data, the data block is stored (written) to a new location on disk to thereby optimize write performance. A write-anywhere file system may initially assume an optimal layout such that the data is substantially contiguously arranged on disks. The optimal disk layout results in efficient access operations, particularly for sequential read operations, directed to the disks. An example of a write-anywhere file system that is configured to operate on a storage system is the Write Anywhere File Layout (WAFL™) file system available from Network Appliance, Inc., Sunnyvale, Calif.

Most RAID implementations enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of redundant information with respect to the striped data. The redundant information, e.g., parity information, enables recovery of data lost when a disk fails. A parity value may be computed by summing (usually modulo 2) data of a particular word size (usually one bit) across a number of similar disks holding different data and then storing the results on an additional similar disk. That is, parity may be computed on vectors 1-bit wide, composed of bits in corresponding positions on each of the disks. When computed on vectors 1-bit wide, the parity can be either the computed sum or its complement; these are referred to as even and odd parity respectively. Addition and subtraction on 1-bit vectors are both equivalent to exclusive-OR (XOR) logical operations. The data is then protected against the loss of any one of the disks, or of any portion of the data on any one of the disks. If the disk storing the parity is lost, the parity can be regenerated from the data. If one of the data disks is lost, the data can be regenerated by adding the contents of the surviving data disks together and then subtracting the result from the stored parity.

Typically, the disks are divided into parity groups, each of which comprises one or more data disks and a parity disk. A parity set is a set of blocks, including several data blocks and one parity block, where the parity block is the XOR of all the data blocks. A parity group is a set of disks from which one or more parity sets are selected. The disk space is divided into stripes, with each stripe containing one block from each disk. The blocks of a stripe are usually at the same locations on each disk in the parity group. Within a stripe, all but one block contains data ("data blocks"), while one block contains parity ("parity block") computed by the XOR of all the data.

As used herein, the term "encoding" means the computation of one or more redundancy values over a predetermined subset of data blocks, whereas the term "decoding" means the reconstruction of one or more data or parity blocks by the same process as the redundancy computation using a subset of data blocks and redundancy values. A typical method for calculating a redundancy value involves computing a parity value by XORing the contents of all the non-redundant blocks in the stripe. If one disk fails in the parity group, the contents of that disk can be decoded (reconstructed) on a spare disk or disks by adding all the contents of the remaining data blocks and subtracting the result from the parity block. Since two's complement addition and subtraction over 1-bit fields are both equivalent to XOR operations, this reconstruction consists of the XOR of all the surviving data and parity blocks. Similarly, if the parity disk is lost, it can be recomputed in the same way from the surviving data.

If the parity blocks are all stored on one disk, thereby providing a single disk that contains all (and only) parity information, a RAID-4 level implementation is provided.

The RAID-4 implementation is conceptually the simplest form of advanced RAID (i.e., more than striping and mirroring) since it fixes the position of the parity information in each RAID group. In particular, a RAID-4 implementation provides protection from single disk errors with a single additional disk, while making it easy to incrementally add data disks to a RAID group.

If the parity blocks are contained within different disks in each stripe, usually in a rotating pattern, then the implementation is RAID-5. Most commercial implementations that use advanced RAID techniques use RAID-5 level implementations, which distribute the parity information. A motivation for choosing a RAID-5 implementation is that, for most read-optimizing file systems, using a RAID-4 implementation would limit write throughput. Such read-optimizing file systems tend to scatter write data across many stripes in the disk array, causing the parity disks to seek for each stripe written. However, a write-anywhere file system, such as the WAFL file system, does not have this issue since it concentrates write data on a few nearby stripes.

While a write-anywhere file system eliminates the write performance degradation normally associated with RAID-4, the fact that one disk is dedicated to parity storage means that it does not participate in read operations, reducing read throughput. Although this effect is insignificant for large RAID group sizes, those group sizes have been decreasing primarily because of two reasons, both of which relate to increasing sizes of disks. Larger disks take longer to reconstruct after failures, increasing the vulnerability of the disk array to a second failure. This can be countered by decreasing the number of disks in the array. Also, for a fixed amount of data, it takes fewer larger disks to hold that data. But this increases the fraction of disks unavailable to service read operations in a RAID-4 configuration. The use of a RAID-4 level implementation may therefore result in significant loss of read operations per second.

When a new disk is added to a full RAID-4 volume, the write anywhere file system tends to direct most of the write data traffic to the new disk, which is where most of the free space is located. A RAID-5 level implementation would do a better job of distributing read and write load across the disks, but it has the disadvantage that the fixed pattern of parity placement makes it difficult to add disks to the array.

Therefore, it is desirable to provide a parity distribution system that enables a storage system to distribute parity evenly, or nearly evenly, among disks of the system.

In addition, it is desirable to provide a parity distribution system that enables a write anywhere file system of a storage system to run with better performance in smaller (RAID group) configurations.

SUMMARY OF THE INVENTION

The present invention overcomes the disadvantages of the prior art by providing a dynamic parity distribution system and technique that distributes parity across disks of an array. The dynamic parity distribution system includes a storage operating system that integrates a file system with a RAID system. In response to a request to store (write) data on the array, the file system determines which disks contain free blocks in a next allocated stripe of the array. There may be multiple blocks within the stripe that do not contain file system data (i.e., unallocated data blocks) and that could potentially store parity (redundant information). One or more of those unallocated data blocks can be assigned to store parity, arbitrarily. According to the dynamic parity distribution technique, the file system determines which

blocks hold parity each time there is a write request to the stripe. The technique alternately allows the RAID system to assign a block to contain parity when each stripe is written.

In the illustrative embodiment, the file system maintains at least one unallocated block per stripe for use by the RAID system. During block allocation, the file system provides an indication to the RAID system of the unallocated block(s) to be used to store parity information. All unallocated blocks on the disks of the array are suitable candidates for file system data or parity. Notably, the unallocated block(s) used to store parity may be located in any disk and the location(s) of the unallocated block(s) can change over time. The file system knows, i.e., maintains information, about the locations of allocated data so that it can leave (reserve) sufficient space for parity in every stripe. The file system illustratively maintains this knowledge through block allocation information data structures.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

FIG. **1** is a schematic block diagram of a storage system that may be advantageously used with the present invention; and

FIG. **2** is a flowchart illustrating a sequence of steps for distributing parity among disks in accordance with a dynamic parity distribution technique of the present invention.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

FIG. **1** is a schematic block diagram of a storage system **100** that may be advantageously used with the present invention. In the illustrative embodiment, the storage system **100** comprises a processor **122**, a memory **124** and a storage adapter **128** interconnected by a system bus **125**. The memory **124** comprises storage locations that are addressable by the processor and adapter for storing software program code and data structures associated with the present invention. The processor and adapter may, in turn, comprise processing elements and/or logic circuitry configured to execute the software code and manipulate the data structures. It will be apparent to those skilled in the art that other processing and memory means, including various computer readable media, may be used for storing and executing program instructions pertaining to the inventive technique described herein.

A storage operating system **150**, portions of which are typically resident in memory and executed by the processing elements, functionally organizes the system **100** by, inter alia, invoking storage operations executed by the storage system. The storage operating system implements a high-level module to logically organize the information as a hierarchical structure of directories, files and blocks on disks of an array. The operating system **150** further implements a storage module that manages the storage and retrieval of the information to and from the disks in accordance with write and read operations. It should be noted that the high-level and storage modules can be implemented in software, hardware, firmware, or a combination thereof.

Specifically, the high-level module may comprise a file system **160** or other module, such as a database, that

allocates storage space for itself in the disk array and that controls the layout of data on that array. In addition, the storage module may comprise a disk array control system or RAID system **170** configured to compute redundant (e.g., parity) information using a redundant storage algorithm and recover from disk failures. The disk array control system ("disk array controller") or RAID system may further compute the redundant information using algebraic and algorithmic calculations in response to the placement of fixed data on the array. It should be noted that the term "RAID system" is synonymous with "disk array control system or disk array controller" and, as such, use of the term RAID system does not imply employment of one of the known RAID techniques. Rather, the RAID system of the invention employs the inventive dynamic parity distribution technique. As described herein, the file system or database makes decisions about where to place data on the array and forwards those decisions to the RAID system.

In the illustrative embodiment, the storage operating system is preferably the NetApp® Data ONTAP™ operating system available from Network Appliance, Inc., Sunnyvale, Calif. that implements a Write Anywhere File Layout (WAFL™) file system having an on-disk format representation that is block-based using, e.g., 4 kilobyte (kB) WAFL blocks. However, it is expressly contemplated that any appropriate storage operating system including, for example, a write in-place file system may be enhanced for use in accordance with the inventive principles described herein. As such, where the term "WAFL" is employed, it should be taken broadly to refer to any storage operating system that is otherwise adaptable to the teachings of this invention.

As used herein, the term "storage operating system" generally refers to the computer-executable code operable to perform a storage function in a storage system, e.g., that manages file semantics and may, in the case of a file server, implement file system semantics and manage data access. In this sense, the ONTAP software is an example of such a storage operating system implemented as a microkernel and including a WAFL layer to implement the WAFL file system semantics and manage data access. The storage operating system can also be implemented as an application program operating over a general-purpose operating system, such as UNIX® or Windows NT®, or as a general-purpose operating system with configurable functionality, which is configured for storage applications as described herein.

The storage adapter **128** cooperates with the storage operating system **150** executing on the system **100** to access information requested by a user (or client). The information may be stored on any type of attached array of writeable storage device media such as video tape, optical, DVD, magnetic tape, bubble memory, electronic random access memory, micro-electro mechanical and any other similar media adapted to store information, including data and parity information. However, as illustratively described herein, the information is preferably stored on the disks **130**, such as HDD and/or DASD, of array **110**. The storage adapter includes input/output (I/O) interface circuitry that couples to the disks over an I/O interconnect arrangement, such as a conventional high-performance, Fibre Channel serial link topology.

Storage of information on array **110** is preferably implemented as one or more storage "volumes" (e.g., VOL**1**-**2** **140**) that comprise a cluster of physical storage disks **130**, defining an overall logical arrangement of disk space. Each volume is generally, although not necessarily, associated with its own file system. The disks within a volume/file system are typically organized as one or more groups, wherein each group is comparable to a RAID group. Most RAID implementations enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate constructing and storing of parity (redundant) information with respect to the striped data.

Specifically, each volume **140** is constructed from an array of physical disks **130** that are divided into blocks, with the blocks being organized into stripes. The disks are organized as groups **132**, **134**, and **136**. Although these groups are comparable to RAID groups, a dynamic parity distribution technique described herein is used within each group. Each stripe in each group has one or more parity blocks, depending on the degree of failure tolerance required of the group. The selection of which disk(s) in each stripe contains parity is not determined by the RAID configuration, as it would be in a conventional RAID-4 or RAID-5 array. Rather, this determination can be made by an external system, such as the file system or array controller that controls the array. The selection of which disks hold parity can be made arbitrarily for each stripe, and can vary from stripe to stripe.

In accordance with the present invention, the dynamic parity distribution system and technique distributes parity across disks of the array. The dynamic parity distribution system includes storage operating system **150** that integrates file system **160** with RAID system **170**. In response to a request to store (write) data on the array, the file system determines which disks contain free blocks in a next allocated stripe of the array. There may be multiple blocks within the stripe that do not contain file system data (i.e., unallocated data blocks) and that could potentially store parity. Note that references to the file system data do not preclude data generated by other high-level modules, such as databases. One or more of those unallocated data blocks can be assigned to store parity, arbitrarily. According to the dynamic parity distribution technique, the file system determines which blocks hold parity each time there is a write request to the stripe. The technique alternately allows the RAID system to assign a block to contain parity when each stripe is written.

In a symmetric parity array, the role of each disk, i.e., whether it stores either data or parity, can vary in each stripe, while maintaining invariants that allow reconstruction from failures to proceed without knowledge of the role each disk block assumed in the array before the failure occurred. Thus symmetric parity, in this context, denotes that the RAID system **170** (or disk array controller such as, e.g., a RAID controller of a RAID array) can reconstruct a lost (failed) disk without knowledge of the role of any disk within the stripe. A typical single redundant storage algorithm, such as single parity, does not require knowledge of the relative positions of the disks in a row. Yet a symmetric double failure-correcting algorithm, such as symmetric row-diagonal (SRD) parity, does require knowledge of the relative positions of the disks in the array, but not of their roles. Furthermore, the algorithmic relationship among all the disks is symmetric. SRD parity is described in co-pending and commonly assigned U.S. Patent Application Serial No. (112056-0141) titled Symmetric Double Failure Correcting Technique for Protecting against Two Disk Failures in a Disk Array, by Peter F. Corbett et al.

The RAID system must "know", i.e., maintain information, about the location of data so that it will not be overwritten; however, the system does not need to know which block contains parity information in order to recon-

struct a failed block. The RAID system simply performs XOR operations on all the other blocks, regardless of content, to reconstruct the data. Notably, the RAID system never needs to know which blocks contain parity; it only needs to know which blocks in a stripe do not contain file system data and that there are one or more such blocks in the stripe. When the RAID system writes new data to a stripe, it can choose any block that does not contain data (an unallocated data block) and place new parity information in it to restore the stripe's parity invariant, e.g., all blocks in the stripe add to zero in response to the XOR operations. The file system or database can make the determination of which block(s) in each stripe to leave available for redundant information (e.g., parity).

With single (row) parity, only one block in a stripe need be chosen to contain a value that sets the sum of the blocks in the stripe to zero. With double parity two blocks in the stripe are chosen to contain values that set the sum of the blocks within row parity sets ("rows") and diagonal parity sets ("diagonals") in the stripe to zero. Other single and double correcting algorithms may also be used advantageously with the invention, as long as they allow any one or more lost blocks to be reconstructed from the surviving blocks in each stripe, independently of whether the lost or surviving blocks contained data or redundant information. Unlike row parity, diagonal parity used in a Row-Diagonal (RD) parity technique is not symmetric, as diagonal parity is not computed for the one diagonal that does not include the diagonal parity disk. Accordingly, the RAID system needs to know of the disk that contains the diagonal parity information in order to reconstruct data using diagonal parity or to compute diagonal parity. The RD parity technique is described in U.S. patent application Ser. No. 10/035,607 titled Row-Diagonal Parity Technique for Enabling Efficient Recovery from Double Failures in a Storage Array, by Peter F. Corbett et al., filed on Dec. 28, 2001.

However, it is possible to utilize an asymmetric redundant storage algorithm, such as RD parity, in such a way as to arbitrarily select any blocks to store data or redundant/parity information in each row. Use of RD parity to compute the redundant information requires information be maintained about the position of each disk in the array. In addition, the asymmetric algorithm requires information about the particular relationship of the contents of each disk to each other disk. Specifically, the contents of some blocks that would typically contain row or diagonal parity may be fixed, setting those blocks to arbitrary data values. RD parity construction or reconstruction algorithms may then be used to determine the contents of the two redundant blocks in each stripe. While such an asymmetric algorithm can be applied in the context of dynamic parity placement, symmetric algorithms have the benefit of being simpler and more convenient to work with.

In a symmetric, double failure-correcting storage algorithm, such as SRD parity, the RAID system generates two disks worth of "redundant" information for storage in an array, wherein the redundant information (e.g., parity) is derived from both diagonal and row parity computation contributions. The RAID system computes the row parity along rows of the array and diagonal parity along diagonals of the array. However, the contents of the redundant parity information disks interact such that neither disk contains purely (solely) diagonal or row parity information; the redundant information is generated using diagonal parity in row parity computations.

A file system **160**, such as the illustrative WAFL file system, typically maintains more information about the location of data on disk than a typical RAID system. The file system knows that subsets of data are allocated and thus contain file system data. The file system illustratively maintains this knowledge through block allocation information data structures, such as an active map **182** and a summary map **184**. Examples of block allocation data structures, such as an active map and a summary map, that may be advantageously used with the present invention are described in U.S. Patent Application Publication No. U.S. 2002/0083037 A1, titled Instant Snapshot and published on Jun. 27, 2002, which application is hereby incorporated by reference. For example, the file system knows that there are certain blocks that contain file system data in a stripe and that cannot change. The only other information about the stripe that the file system needs to know is that all the blocks in the stripe add to zero when XOR operations are performed thereon. Therefore, any of the blocks that do not contain data (unallocated data blocks) can be modified to ensure that the blocks add to zero.

Since successful reconstruction is independent of the disk(s) chosen to hold parity information, parity disk(s) can be chosen from among free blocks by either the file system or RAID system and selection can vary from stripe to stripe, as in a RAID-5 level implementation. This, in turn, allows the file system and/or RAID system to render dynamic decisions as to where to place (distribute) parity information in either a row parity stripe arrangement or a double failure correcting parity array.

Dynamic parity distribution is based on the above-described observation about the relationship between the file system and RAID system, and, in the illustrative embodiment described herein, on the symmetric behavior of the parity algorithm. According to the inventive technique, the file system **160** maintains at least one unallocated block (two for a double failure correcting algorithm) per stripe for use by the RAID system **170**. During block allocation, the file system provides an indication to the RAID system of the unallocated block(s) to be used to contain parity information. All unallocated blocks on the disks of the array are suitable candidates for file system data or parity. Notably, the unallocated block(s) used to store parity may be located in any disk and the location(s) of the unallocated block(s) can change over time. Moreover, all blocks in a RAID group are available for potential allocation, since parity is not held in fixed locations. In practice, this means that all blocks, including those that were "hidden" in the parity disk are available to the file system **160** for allocation in volume block number space. The file system has knowledge of the locations of allocated data so that it can leave (reserve) sufficient space for parity in every stripe.

FIG. **2** is a flowchart illustrating a sequence of steps for distributing parity among disks of an array in accordance with the dynamic parity distribution technique of the present invention. According to the technique, the file system **160** indicates to the RAID system **170** which block(s) in a next allocated stripe holds parity each time there is a write operation (request) involving write data to the stripe. The sequence starts at Step **200** and proceeds to Step **202** where the file system (high-level module) determines which disks contain free blocks in the stripe in response to the write request. The stripe will contain at least one unallocated block, which is the parity block, and one or more unallocated blocks that are freed data blocks. All blocks contribute to, e.g., even parity, so the parity block(s) and the freed data blocks are all equivalent. In Step **204**, the file system reserves as many free blocks as required by the redundant

storage algorithm to store parity, with the remaining unallocated blocks used to store data.

In Step **206**, write allocation code ("write allocator **165**") of the file system arranges (i.e., "lays out") the write data for storage on the disks in the stripe. The RAID system provides topology information to the file system about the disks of each group **132-136** that allows the write allocator to render optimal and correct write allocation decisions. Write allocation is directed to one group at a time to enable writing of full stripes across the disks of the group. In Step **208**, the file system provides an indication of the reserved block(s) to the RAID system (storage module) via, e.g., a write request message and, in Step **210**, the RAID system assigns parity to the reserved block(s). In Step **212**, the RAID system provides the parity information (and write data) to a disk driver system (not shown) for storage at the assigned block locations on the disk. The sequence then ends at Step **214**.

Note that in a preferred embodiment of the inventive dynamic parity distribution technique, the file system **160** simply tracks the locations of allocated file data blocks. The RAID system **170** loads the reserved block(s) with parity information, but the file system does not need to know which block(s) contain parity. The RAID system knows which blocks are being written, so that it manages changes in the parity relationships. However, in an alternate embodiment, the file system may track the reserved blocks using parity reservation information maintained in a separate parity block map structure **188**. The parity block map **188** requires constant update whenever a parity block is reassigned.

Note also that the symmetry property with respect to reconstruction is very helpful, as it makes it possible to use the same algorithm to compute the lost data or redundant information, regardless of which disk(s) fail. However, even in the case where an asymmetric algorithm is used to correct one or more failures, knowledge of the position of the redundant information in each stripe is not required to fully reconstruct the lost contents of the stripe, as long as the position and role of each disk are known with respect to the algorithm used.

Dynamic parity distribution may be employed to compensate for uneven distribution of write data across the disks **130** of array **300** to thereby evenly distribute (i.e., balance) the data access load across the disks. As the file system attempts to fully populate each stripe with data, the RAID system **170** can reduce the data access load by distributing parity to a "hot" disk, i.e., a disk that is more heavily utilized than other disks. As noted, any unallocated block can be used to contain parity. At the point of writing to a stripe, the file system chooses one or two of the unallocated blocks to hold parity from among the disks that contain the most data to thereby reassign future read workload away from a heavily utilized (i.e., "hot") disk. Here, the file system may maintain a counter **186** for each disk **130** to keep track of the amount of data that has been newly allocated on the disk. Each counter may be stateless; when the operating system reboots, each counter is set to zero. This arrangement tends to equalize the amount of data on each disk over time.

The file system **160** may render further parity distribution decisions, depending on the situation. For example, assume a new disk is added to a RAID group. Before the disk is added to the group, it is preferably zeroed so that it is neutral with respect to each stripe's parity invariant. Once the disk is added, the file system has a new set of free blocks at its disposal. The file system may proportionally choose to use the free block(s) or the previous parity block(s) for parity. This, in turn, allows new write data to be allocated in the old parity locations on each stripe, thus distributing any future read load across the array.

Dynamic parity distribution has interesting implications operating in degraded mode, e.g., after one or more disks have failed. When running in a degraded state, it may be possible to only reconstruct the lost data blocks, relocating them on the surviving disks where those disks previously held parity. This does not restore data protection to those stripes, but it does improve the performance of subsequent read accesses. Operating in such a degraded state requires some interaction between the RAID system **170** and a "client" of the RAID system (such as the file system **160**) or requires re-identification of the relocated blocks at the interface to the RAID system. Otherwise, high-level modules of the storage operating system **150** would not be able to locate those relocated data blocks.

There are various "degrees of freedom" that can be exploited by the file system using the dynamic distribution technique. For example, the dynamic distribution technique may apply to an array where there are multiple disk sizes in the same group **132-136**. If parity were evenly distributed across the disks, the larger disks would realize more load simply because they contain more data. Dynamic distribution can also use disk size as weighting for the parity distribution system so that the blocks available for data distribution are more evenly allocated across the group.

The dynamic parity distribution technique also allows the performance of various other arrangements, such as biasing parity distribution based on the actual measured read load of every disk. If data is mostly read, then biasing to even the read load across all of the disks may be near optimal. In most cases, biasing is employed to balance the total load across all disks, including the read and write load. This can be accomplished by taking advantage of the average behavior across a large data set and simply balancing the number of data blocks across the disks. Alternatively, an algorithm for biasing parity distribution involves adding for allocated data and subtracting for unallocated blocks, while changing the amounts added and subtracted to bias data for storage on one disk or another of the array. This alternate embodiment includes the use of counters that are maintained close to a predetermined value (e.g., zero) to thereby determine on which disk to store the next data blocks.

For example, a counter for a disk is incremented each time a data block is allocated to the disk where there previously was no data block. Similarly, the counter is decremented whenever a data block is freed or parity block is allocated to the disk. The ratio of the amounts of increment to decrement for each block allocation or de-allocation determines the ratio of data to parity and free blocks on each disk. Keeping the counter close to zero keeps the ratio of data to non-data blocks close to the desired value. This technique can be used to balance the load across all disks, even if the disks are of different sizes.

Biasing with respect to balancing data is handled differently where a group has fewer, but larger disks. Essentially, this enables the file system to balance the parity information on the larger portions of disks within a group **132-136**. Balancing of information is done on a stripe-by-stripe basis, with the dynamic distribution technique being used to bias data or parity in such a way as to balance the read load even though the disks are of different sizes. In the illustrative embodiment, the file system may implement a balancing algorithm to render balancing decisions when determining the block locations of the data and parity information on the disks. The illustrative balancing algorithm includes factors

such as the different sizes of the disk/stripes, measurement of I/O operations to each disk and adjusting the I/O rates accordingly, and adjusting for the speed of each disk to thereby allow utilization of disks having different speeds.

The parity distribution technique described herein is particularly useful for systems having fewer disks yet that want to utilize all read operations per second (ops) that are available from those disks. Performance of smaller arrays is bounded by the ops that are achievable from disks (disk-bound). Yet even in large arrays where disks get larger, because of reconstruction times, the tendency is to reduce the number of disks per group **132-136**. This results in an increase in redundancy overhead (the percentage of disks in a group devoted to redundancy increases). Therefore, it is desirable to take advantage of the read ops available in those redundant disks. Another advantage of the parity distribution technique is that reconstruction and/or recovery occurs "blindly" (i.e., without knowing the roles of the disks). The dynamic parity distribution system and technique applies to single disk failure correction and can be extended to apply to double (or greater) disk loss protection.

Dynamic parity distribution may be advantageously used with arrays having low numbers of large disks, since the technique balances data across the array. Using larger disks is required to get reasonable capacity, but that also means using smaller groups **132-136** to limit reconstruction time. If a 14-disk configuration uses two groups and one spare, then a relatively substantial percentage (e.g., over 20%) of the disks are unavailable for use in storing or retrieving data. Configurations with eight disks are even worse, e.g., one spare and one parity disk amount to 25% overhead. Dynamic parity distribution could make the two parity disks and, potentially, the spare disk in the 14-disk configuration available for data. This overhead reduces the array's effective read capacity.

It will be understood to those skilled in the art that the inventive technique described herein may apply to any type of special-purpose (e.g., file server, filer or multi-protocol storage appliance) or general-purpose computer, including a standalone computer or portion thereof, embodied as or including a storage system **100**. An example of a multi-protocol storage appliance that may be advantageously used with the present invention is described in U.S. patent application Ser. No. 10/215,917 titled, Multi-Protocol Storage Appliance that provides Integrated Support for File and Block Access Protocols, filed on Aug. 8, 2002. Moreover, the teachings of this invention can be adapted to a variety of storage system architectures including, but not limited to, a network-attached storage environment, a storage area network and disk assembly directly-attached to a client or host computer. The term "storage system" should therefore be taken broadly to include such arrangements in addition to any subsystems configured to perform a storage function and associated with other equipment or systems.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For instance, it is expressly contemplated that the teachings of this invention can be implemented as software, including a computer-readable medium having program instructions executing on a computer, hardware, firmware, or a combination thereof. Accordingly this description is to be taken only by way of example and not to otherwise limit the scope of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A system adapted to distribute redundant information across disks of an array, the system comprising:

a storage operating system configured to invoke storage operations executed by a storage system, the storage operating system further configured to manage storage of information, including the redundant information and data, on blocks of the disks in response to disk access operations, the storage operating system including a storage module adapted to compute the redundant information in response to a layout of the data in stripes across the disks, the storage operating system maintaining at least one unallocated block per stripe for use by the storage module to store the computed redundant information, wherein the at least one unallocated block used to store the redundant information is located in any disk and wherein the location of the at least one unallocated block used to store the redundant information is dynamically allocated in a non-fixed pattern by the storage module before each write request is completed for each stripe.

2. The system of claim **1** wherein the storage module is a disk array controller configured to compute the redundant information and reconstruct blocks lost due to failure of one or more of the disks.

3. The system of claim **1** wherein the storage module is a RAID system configured to compute the redundant information and reconstruct blocks lost due to failure of one or more of the disks.

4. The system of claim **3** wherein the storage operating system is further configured to implement a high-level module that maintains information about locations of the data on the disks.

5. The system of claim **4** wherein the high-level module is a file system or database adapted to control layout of the data on the disks.

6. The system of claim **5** wherein the storage operating system integrates the file system or database with the RAID system.

7. The system of claim **6** wherein the file system or database is configured to determine block locations of the data on the disks and the RAID system is configured to determine the block locations of the redundant information on the disks.

8. The system of claim **6** wherein the file system or database is configured to determine block locations of the data and the redundant information on the disks.

9. The system of claim **8** wherein the file system or database renders balancing decisions to determine the block locations of the data and the redundant information on the disks.

10. The system of claim **9** wherein the balancing decisions comprises one of different sizes of disks, different speeds of disks, and whether a disk is more heavily utilized than other disks.

11. The system of claim **8** further comprising block allocation map structures used by the file system to determine the block locations of the data and the redundant information on the disks.

12. The system of claim **11** wherein the redundant information is parity.

13. The system of claim **1** wherein the storage module selects the at least one unallocated block to store the

redundant information and wherein the storage module computes the redundant information using a redundant storage algorithm.

14. The system of claim **13** wherein the selection of the at least one unallocated block to store redundant information is independent of the redundant storage algorithm.

15. The system of claim **14** wherein the redundant storage algorithm is an asymmetric algorithm.

16. The system of claim **14** wherein the redundant storage algorithm is a symmetric algorithm.

17. The system of claim **16** wherein the redundant information is parity.

18. The system of claim **13** wherein the at least one unallocated block used to store the redundant information comprises two or more unallocated blocks used to store the redundant information.

19. The system of claim **18** wherein the selection of the unallocated blocks to store redundant information is independent of the redundant storage algorithm used to compute the redundant information.

20. The system of claim **19** wherein the redundant storage algorithm depends on positions of the blocks in the array.

21. The system of claim **20** wherein the redundant storage algorithm is a symmetric and wherein the redundant information is parity.

22. The system of claim **20** wherein the redundant storage algorithm is an asymmetric algorithm and wherein the redundant information is parity.

23. The system of claim **1**, wherein the non-fixed pattern is created by the redundant information being stored in any block remaining after the data is allocated to blocks of the stripe.

24. A method for distributing redundant information across disks of an array, comprising:

dividing each disk into blocks, the blocks being organized into stripes such that each stripe contains one block from each disk;

selecting one or more blocks in a stripe layout of the stripe not used to contain data to contain the redundant information, wherein the selected one or more blocks used to contain the redundant information are located in any disk and wherein the location of the selected one or more blocks used to contain the redundant information is dynamically allocated in a non-fixed pattern before each write request for each stripe is completed;

computing the redundant information based on contents of all other blocks in the stripe layout, regardless of whether the blocks contain data; and

writing the data in one or more allocated blocks and the redundant information in the one or more selected blocks of the stripe that is written across the disks of the array.

25. The method of claim **24** further comprising the step of determining which block in a stripe contains redundant information each time there is a write request to the stripe.

26. The method of claim **25** further comprising the step of assigning a block to contain redundant information when each stripe is written.

27. The method of claim **26** wherein the step of determining is performed by a high-level module of a storage system and wherein the steps of computing and assigning are performed by a storage module of the storage system.

28. The method of claim **27** further comprising:

maintaining, by the high-level module, at least one unallocated block per stripe for use by the storage module; and

providing an indication from the high-level module to the storage module of the unallocated block to contain parity.

29. The method of claim **28** further comprising the step of reconstructing, using the storage module, a block that is lost due to failure of a disk.

30. The method of claim **27** wherein the high-level module is a file system and wherein the storage module is one of an array controller and a RAID system.

31. The method of claim **30** wherein the step of computing comprises computing the redundant information in response to the stripe layout of the data in stripes across the disks.

32. The method of claim **31** wherein the step of computing further comprises computing the redundant information using algebraic and algorithmic calculations in response to the placement of the data on the array.

33. Apparatus for distributing redundant information across disks of an array, the apparatus comprising:

means for dividing each disk into stripes, with each stripe containing one block from each disk;

means for selecting one or more blocks in a stripe layout of the stripe not used to contain data to contain redundant information, wherein the selected one or more blocks used to contain the redundant information are located in any disk and wherein the location of the selected one or more blocks used to contain the redundant information is dynamically allocated in a non-fixed pattern before each write request for each stripe is completed;

means for computing the redundant information based on contents of all other blocks in the stripe layout, regardless of whether the blocks contain data; and

means for writing the data in one or more allocated blocks and the redundant information in the one or more selected blocks of the stripe that is written across the disks of the array.

34. The apparatus of claim **33** further comprising means for determining which block or blocks in a stripe holds redundant information each time there is a write operation to the stripe.

35. A computer readable medium containing executable program instructions for distributing parity across disks of an array, the executable instructions comprising one or more program instructions for:

dividing each disk into stripes, with each stripe containing one block from each disk;

selecting one or more blocks in a stripe layout of the stripe not used to contain data to contain parity, wherein the selected one or more blocks used to contain the parity are located in any disk and wherein the location of the selected one or more block used to contain the parity is dynamically allocated in a non-fixed pattern before each write request for each stripe is completed;

computing the parity based on contents of all other blocks in the stripe, regardless of whether the blocks contain data; and

writing the data in one or more allocated blocks and the redundant information in the one or more selected blocks of the stripe that is written across the disks of the array.

36. A method for distributing redundant information across disks of an away with a plurality of blocks on each disk, comprising:

determining which blocks are unallocated in a stripe across the disks;

reserving unallocated blocks for storing the redundant information in one or more reserved unallocated blocks;

arranging data in the stripe for the data to be stored in one or more allocated blocks across the disks of the array;

assigning the redundant information in a non-fixed pattern to the one or more reserved unallocated blocks; and

writing the data in the allocated blocks and the redundant information in the one or more reserved unallocated blocks as the stripe across the disks of the array.

37. The method of claim **36**, further comprising:

storing parity information as the redundant information in the one or more reserved unallocated blocks.

38. The method of claim **36**, further comprising:

adding a disk to the array;

storing a second stripe across the array by determining one or more unallocated blocks across the array including the added disk, and writing the data to allocated blocks and the redundant information to the one or more unallocated blocks of the second stripe.

39. An apparatus for distributing redundant information across disks of an array with a plurality of blocks on each disk, comprising:

means for determining which blocks are unallocated in a stripe across the disks;

means for reserving unallocated blocks for storing the redundant information in one or more reserved unallocated blocks;

means for arranging data in the stripe for the data to be stored in one or more allocated blocks across the disks of the array;

means for assigning the redundant information in a non-fixed pattern to the one or more reserved unallocated block; and

means for writing the data in the allocated blocks and the redundant information in the one or more reserved unallocated blocks as the stripe across the disks of the array.

40. The apparatus of claim **39**, further comprising:

means for storing parity information as the redundant information in the one or more reserved unallocated blocks.

41. The apparatus of claim **39**, further comprising:

means for adding a disk to the array;

means for storing a second stripe across the array by determining one or more unallocated blocks across the array including the added disk, and writing the data to allocated blocks and the redundant information to the one or more unallocated blocks of the second stripe.

42. A system for distributing redundant information across disks of an array with a plurality of blocks on each disk, comprising:

a storage operating system configured to invoke storage operations executed by a storage system, the storage operating system further configured (i) to determine which blocks are unallocated in a stripe across the disks, (ii) to reserve unallocated blocks for storing the redundant information in one or more reserved unallocated blocks, (iii) to arrange data in the stripe for the data to be stored in one or more allocated blocks across the disks of the array, (iv) to assign the redundant information in a non-fixed pattern to the one or more reserved unallocated block, and (iv) to write the data in the allocated blocks and the redundant information in the one or more reserved unallocated blocks as the stripe across the disks of the array.

43. The system of claim **42**, wherein the redundant information is parity information.

44. A computer readable medium containing executable program instructions for distributing parity across disks of an array, the executable instructions comprising one or more program instructions for:

determining which blocks are unallocated in a stripe across the disks, where each disk has a plurality of blocks;

reserving unallocated blocks for storing the redundant information in one or more reserved unallocated blocks;

arranging data in the stripe for the data to be stored in one or more allocated blocks across the disks of the array; and

assigning the redundant information in a non-fixed pattern to the one or more reserved unallocated block; and

writing the data in the allocated blocks and the redundant information in the one or more reserved unallocated blocks as the stripe across the disks of the array.

45. A method for distributing redundant information across disks of an array with a plurality of blocks on each disk, comprising:

allocating data to blocks in a stripe layout, where the stripe layout represents a stripe across the disks of the array;

reserving unallocated blocks for storing the redundant information in one or more reserved unallocated blocks;

assigning the redundant information to the one or more reserved unallocated blocks in a non-fixed pattern; and

writing the data in the allocated blocks and the redundant information in the one or more reserved unallocated blocks as the stripe across the disks of the array.

46. The method of claim **45**, further comprising:

storing parity information as the redundant information in the one or more reserved unallocated blocks.

47. The method of claim **45**, further comprising:

adding a disk to the array;

storing a second stripe across the array by determining one or more unallocated blocks across the array including the added disk, and writing the data to allocated blocks and the redundant information to the one or more unallocated blocks of the second stripe.

48. The method of claim **45**, further comprising:

computing the redundant information based on contents of other blocks in the stripe layout, regardless of whether the blocks contain data or are empty.

49. The method of claim **48**, wherein the step of computing uses a symmetric algorithm and wherein the redundant information is parity.

50. The method of claim **48**, wherein the step of computing uses an asymmetric algorithm and wherein the redundant information is parity.

51. The method of claim **45**, wherein the non-fixed pattern is created by the redundant information being stored in any block remaining after the data is allocated to blocks of the stripe.

* * * * *

PROOF OF SERVICE

I declare that I am employed with the law firm of Weil, Gotshal & Manges LLP, whose address is 201 Redwood Shores Parkway, Redwood Shores, California 94065-1175 (hereinafter "WGM"). I am not a party to the within cause, and I am over the age of eighteen years. I further declare that on May 19, 2008, I served a copy of:

NETAPP'S ANSWER TO SUN MICROSYSTEMS, INC.'S COMPLAINT FOR PATENT INFRINGEMENT

☐ **BY U.S. MAIL** by placing a true copy thereof enclosed in a sealed envelope with postage thereon fully prepaid, addressed as follows, for collection and mailing at WGM in accordance with WGM's ordinary business practices. I am readily familiar with WGM's practice for collection and processing of mail, and know that in the ordinary course of WGM's business practice that the document(s) described above will be deposited with the U.S. Postal Service on the same date as sworn to below.

☐ **BY FACSIMILE** by sending a true copy from WGM's facsimile transmission telephone number of 650-802-3100 to the fax number(s) set forth in the service list below. The transmission was reported as complete and without error. The transmission report was properly issued by the transmitting facsimile machine.

☒ **BY ELECTRONIC SERVICE** by electronically mailing a true and correct copy through WGM's electronic mail system to the email address(es) set forth in the service list below. Also, the email addresses registered with the court ECF system each received a notice of electronic filing upon completion of the electronic filing.

☐ **BY OVERNIGHT DELIVERY** by placing a true copy thereof enclosed in a sealed envelope with overnight delivery fees provided for, addressed as follows, for collection by Federal Express at WGM in accordance with WGM's ordinary business practices. I am readily familiar with WGM's practice for collection and processing of correspondence for overnight delivery and know that in the ordinary course of WGM's business practice the document(s) described above will be deposited by an employee or agent of WGM in a box or other facility regularly maintained by Federal Express for collection on the same day that the document(s) are placed at WGM.

☐ **BY PERSONAL SERVICE** by placing a true copy thereof enclosed in a sealed envelope to be delivered by messenger to the offices of the addressee(s) (and left with an employee or person in charge of addressee's office), as stated below, during ordinary business hours on May 19, 2008.

Mark D. Fowler
mark.fowler@dlapiper.com
David L. Alberti
david.alberti@dlapiper.com
Christine K. Corbett
christine.corbett@dlapiper.com
Yakov M. Zolotorev
yakov.zolotorev@dlapiper.com
Carrie L. Williamson
carrie.williamson@dlapiper.com
DLA Piper US LLP
2000 University Avenue
East Palo Alto, CA 94303
Tel: (650) 833-2000

☐ **by U.S. Mail**
☐ **by fax**
☒ **by email**
☐ **by overnight delivery**
☐ **by hand**

Fax: (650) 833-2001

Executed on May 19, 2008 at Redwood Shores, California. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

/s/
Jeffrey G. Homrig